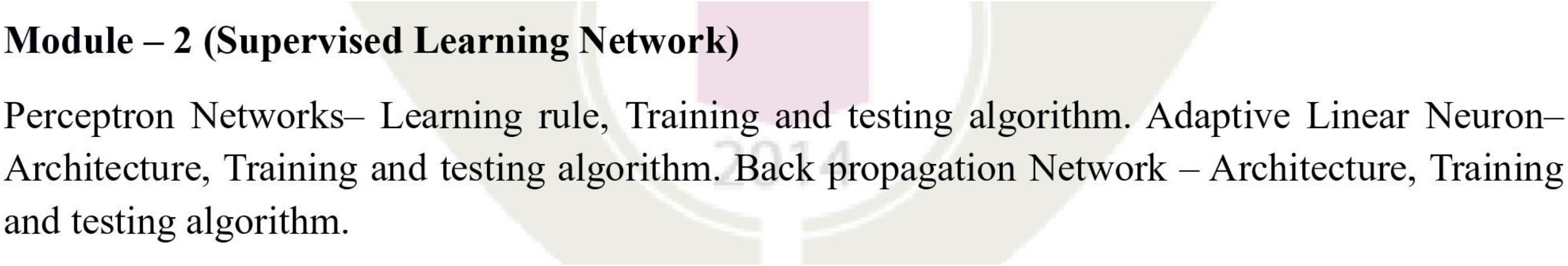
MODULE-2

SUPERVISED LEARNING NETWORK



Module – 2 (Supervised Learning Network)

Perceptron Networks– Learning rule, Training and testing algorithm. Adaptive Linear Neuron– Architecture, Training and testing algorithm. Back propagation Network – Architecture, Training and testing algorithm.

LAYERS IN NEURAL NETWORK

- **The input layer:**
  - Introduces input values into the network.
  - No activation function or other processing.

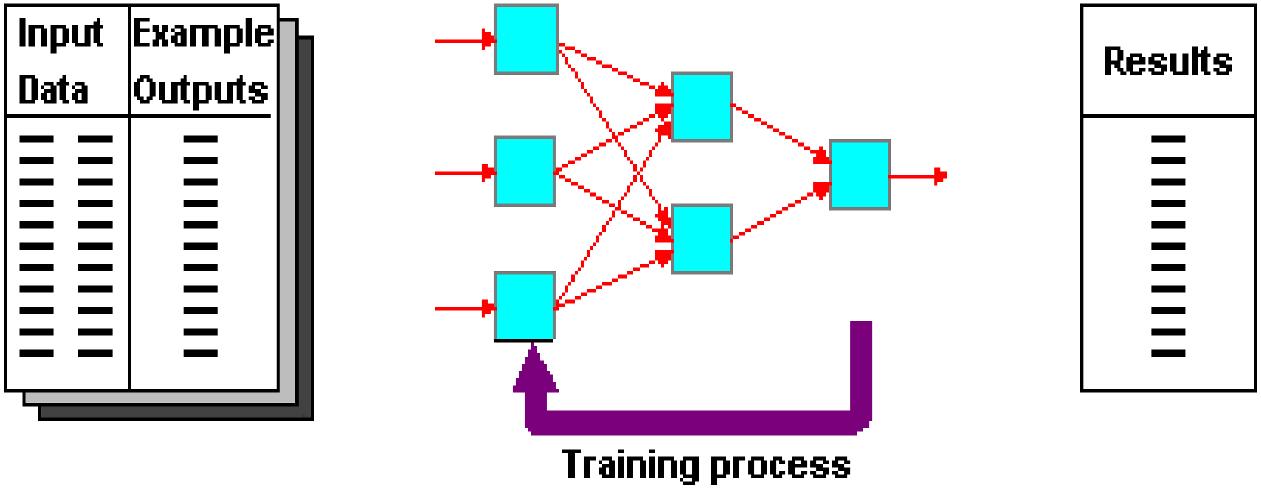
- **The hidden layer(s):**
  - Performs classification of features.
  - Two hidden layers are sufficient to solve any problem.
  - Features imply more layers may be better.

- **The output layer:**
  - Functionally is just like the hidden layers.
  - Outputs are passed on to the world outside the neural network.

DEFINITION OF SUPERVISED LEARNING NETWORKS

- Training and test data sets

- Training set; input & target are specified



PERCEPTRON NETWORKS

- Single layer feed forward network
- First neural network learning model in the 1960’s
- Simple and limited (single layer models)
- Basic concepts are similar for multi-layer models so this is a good learning tool
- Still used in many current applications (modems, etc.)
- Called Simple perceptron
- Discovered by Block in 1962

PERCEPTRON NETWORKS

✓Key points to be noted in perceptron network are

- ❖Consist of 3 units
  - ❑***Sensory unit(input unit)***
  - ❑***Associator Unit(hidden unit)***
  - ❑***Response unit(output unit)***

- ❖***Sensory units*** are connected to ***associator unit*** with a fixed weight having values 1, 0 or -1 which are assigned at random
- ❖Binary activation functions are used in the sensory unit and associator unit
- ❖The ***response unit*** has an activation of 1,0 or -1
- ❖The binary step with fixed threshold $\boldsymbol{\theta}$ is used as an activation for associator
- ❖The output signal that are send from the associator to response are only binary

- The output of perceptron network is given by

$$Y= f(y_{in})$$

- Where $f(y_{in})$ is the activation function and is defined as

$$f(y_{in})= \begin{cases} 1 & \text{If } y_{in} > \theta \\ 0 & \text{If } - \theta <= y_{in} <= \theta \\ -1 & \text{If } y_{in} < - \theta \end{cases}$$

- The perceptron learning rule is used in the weight updation between the associator unit and the response unit
- For each training input, the net will calculate the response and it will determine whether or not an error has occurred
- The error calculations are based on the comparison of values for targets with those of calculated output

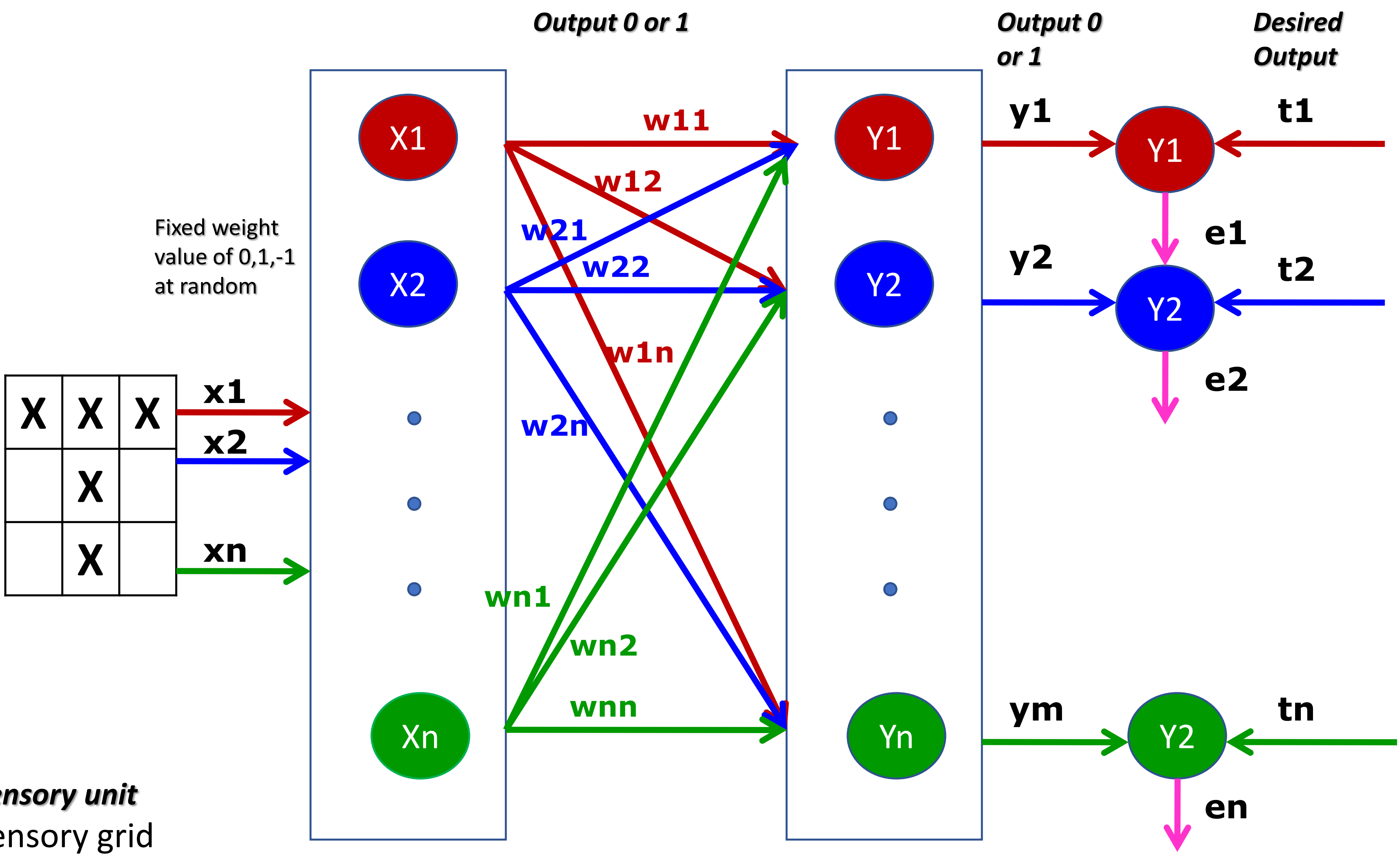
- The weights on the connection from the unit that send the non zero signal will get adjusted suitably
- Weight will be adjusted on the basis of the learning rule if an error has occurred for a particular training pattern

$$wi(new)=wi(old)+\alpha\, t\, xi$$

$$b(new)=b(old)+\alpha\, t$$

- If no error occurred then no weight updation and hence the training process may be stopped
- In the above equation the target value ‘t’ is +1 or -1
- α is the learning rate

- ln general, these learning rules begin with an initial guess at the weight values and
- then successive adjustments are made on the basis of the evaluation of an objective function.
- Eventually, the learning rules reach a near optimal or optimal solution in a finite number of steps.



Fixed weight value of 0,1,-1 at random

X	X	X
	X	
	X	

Sensory unit

Sensory grid representing any pattern

Associator Unit

Response Unit

✓ *Sensory unit*

- ➢can be a 2D matrix of 400 photo detectors
- ➢These detectors provide a binary electrical signal if the input signal is found to exceed a certain value of threshold
- ➢Also these detectors are connected randomly with the associator unit

✓ *Associator unit*

- ➢consist of a set of sub circuits called ***feature predicates***
  - ➢Hardwired to detect the specific feature of a pattern
- ➢For a particular feature each predicate is examined with a few or all of the responses of sensory unit
- ➢The result from these predicate unit is also binary

✓ *Response Unit*

- ➢Contain pattern recognizer or perceptron
- ➢The weight present in the input layer are all fixed
- ➢While weight in the response layer are trainable

LEARNING ALGORITHM

- **Epoch** : Presentation of the entire training set to the neural network.

- In the case of the AND function, an epoch consists of four sets of inputs being presented to the network (i.e. [0,0], [0,1], [1,0], [1,1]).

- **Error**: The error value is the amount by which the value output by the network differs from the target value. For example, if we required the network to output 0 and it outputs 1, then Error = -1.

PERCEPTRON LEARNING RULE

- ✓In learning rule, the learning signal is the difference between the desired and actual response of the neuron
- ✓Learning rule explained as follows
  - Consider a finite 'n' number of input training vectors, with their associated target values x(n) and t(n), where n ranges from 1 to n
  - The target is either +1 or -1
  - The output y is obtained on the basis of the net input calculated and activation function being applied over the net input

$$y=f(y_{in})=\begin{cases} 1 & \text{If } y_{In} > \theta \\ 0 & \text{If } -\theta <= y_{In} <= \theta \\ -1 & \text{If } y_{in} < -\theta \end{cases}$$

- Weight updation in case of perceptron learning is as shown

  If $y \neq t$ then

  $$w(new) = w(old) + \alpha\, t\, x \quad (\alpha \text{ learning rate})$$

  Else we have

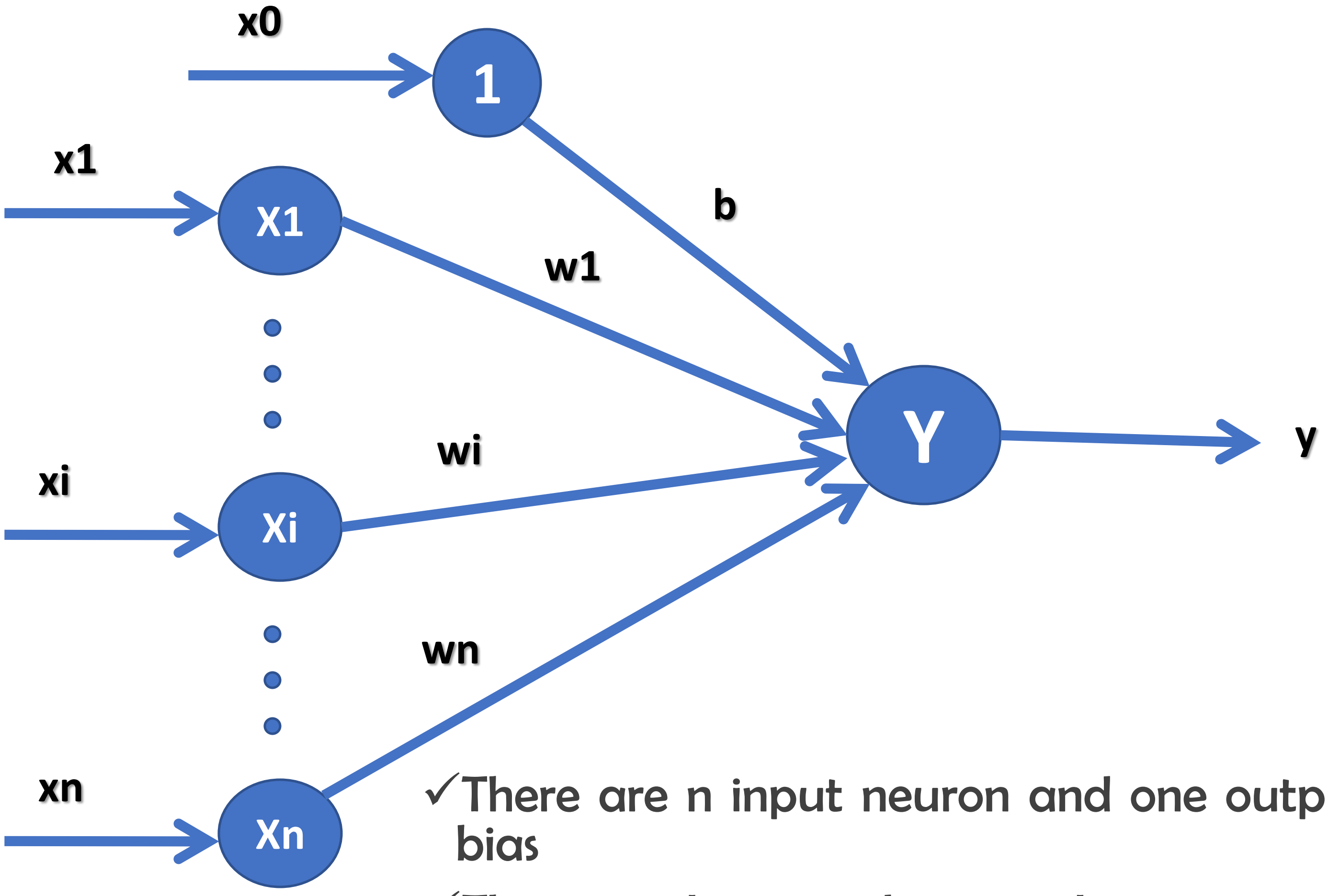
  $$w(new) = w(old)$$

- Weight can be intialized to any values in this method
- Perceptron rule convergence theorom states that

  "If there is a weight vector W , such that f (x (n) W)= t(n) , for all n, then for any starting vector W1, the perceptron learning rule converges to a weight vector that gives the correct response for all training patterns and this learning take place within a finite number of steps provided that the solution exist"

ARCHITECTURE

- ✓The output obtained from the associator unit is a binary vector and hence that the output is taken as input signal to the response unit
- ✓Here the weight between the sensory unit and associator unit are fixed
- ✓Weight between the associator unit and response unit output unit can be adjusted
- ✓As a result the discussion of the network is limited to a single portion
- ✓Thus associator unit behaves like input unit



- ✓There are n input neuron and one output neuron and a bias
- ✓The input layer and output layer are connected through the direct communication link which is associated with weight
- ✓The goal of perceptron network is to classify the input pattern as the member or not a member of a particular class

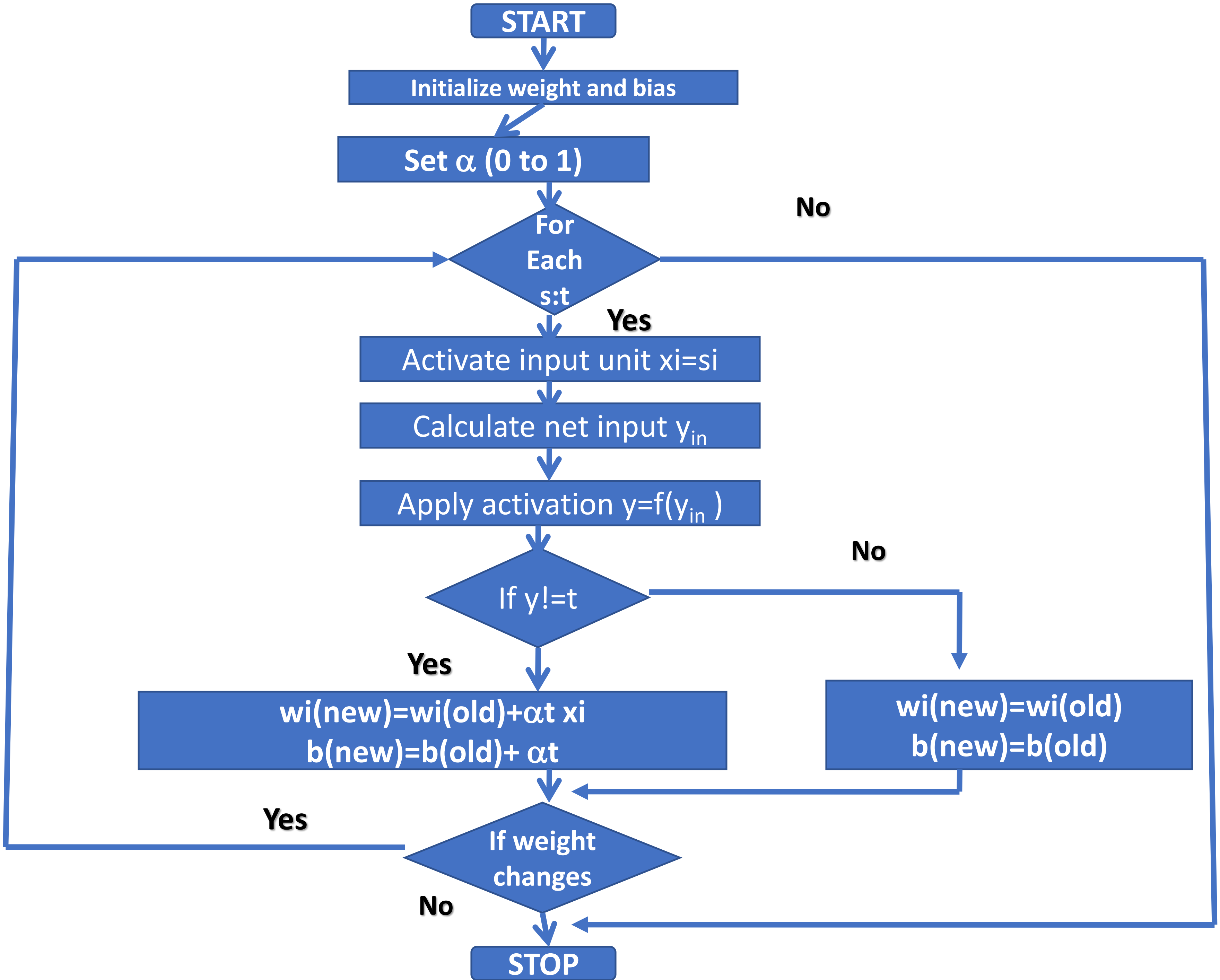
TRAINING ALGORITHM

- Adjust neural network weights to map inputs to outputs.

- Use a set of sample patterns where the desired output (given the inputs presented) is known.

- The purpose is to learn to
  - Recognize features which are common to good and bad

FLOW CHART OF A TRAINING PROCESS


START
Initialize weight and bias
Set α (0 to 1)
For Each s:t
No
Yes
Activate input unit xi=si
Calculate net input y_{in}
Apply activation $y=f(y_{in})$
If y!=t
No
Yes
wi(new)=wi(old)+αt xi
b(new)=b(old)+ αt
wi(new)=wi(old)
b(new)=b(old)
Yes
If weight changes
No
STOP

PERCEPTRON TRAINING ALGORITHM FOR SINGLE OUTPUT CLASS

Step 0: Initialize the weight and bias (to zero). Also initialize learning rate α ($0 < \alpha <= 1$) For simplicity α set to 1

Step 1: Perform step 2- 6 until the final stopping condition is false

Step 2: Perform step 3-5 for each training pair indicated by s:t

Step 3: Input layer containing the input unit is applied with identity activation functions

$$x_i = s_i$$

Step 4: Calculate the output of the network. To do so first obtain the net input

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

'n' is the number of neurons in the layer

Then apply the activation function over net input to get the output

$$y = f(y_{in}) = \begin{cases} 1 & \text{If } y_{in} > \theta \\ 0 & \text{If } -\theta <= y_{in} <= \theta \\ -1 & \text{If } y_{in} < -\theta \end{cases}$$

Step 5: Weight and bias adjustment: Compare values of actual and desired output

If $y \neq t$ then $wi(new)=wi(old)+\alpha\, t\, xi$

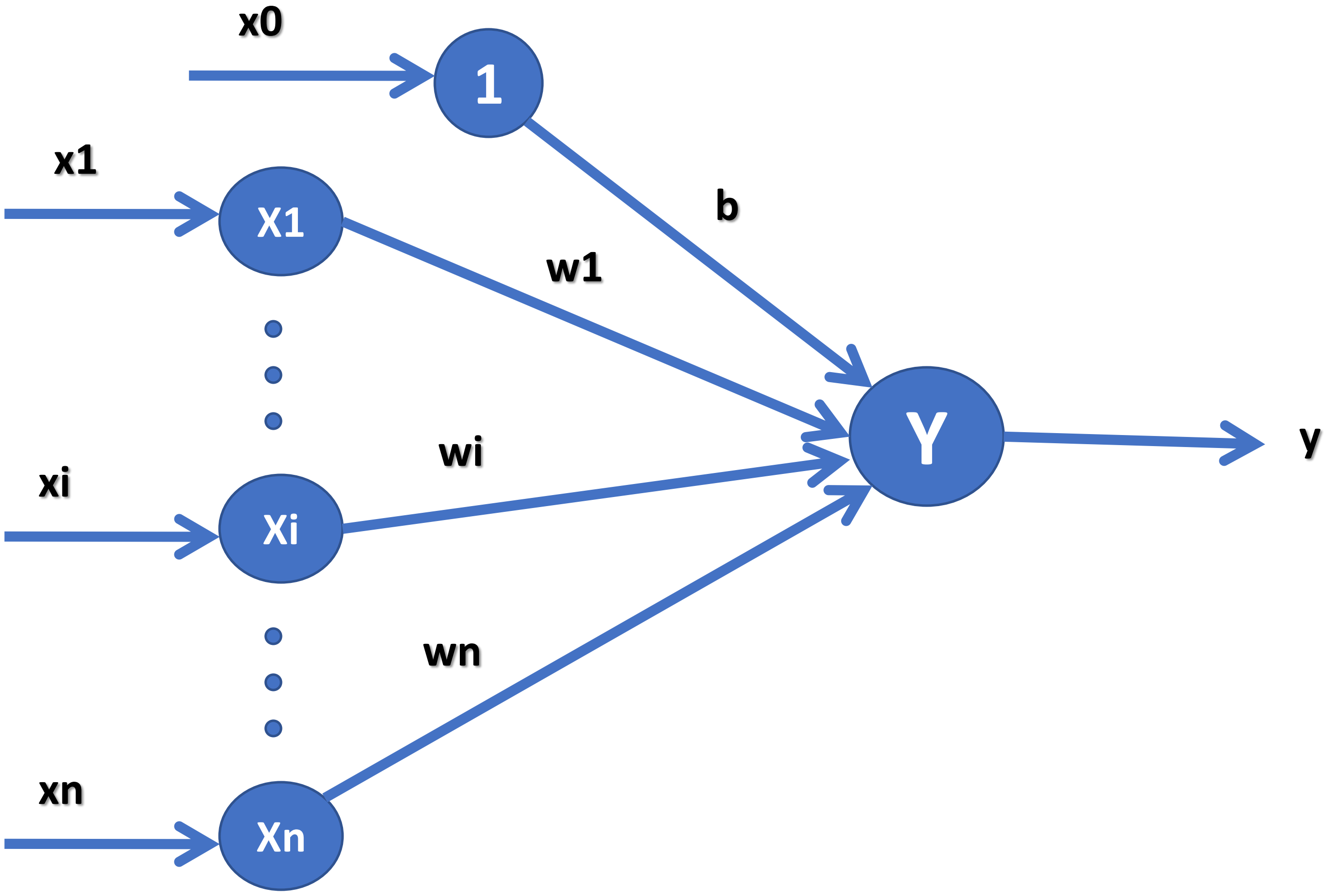
$b(new)=b(old)+ \alpha t$

else $wi(new)=wi(old)$

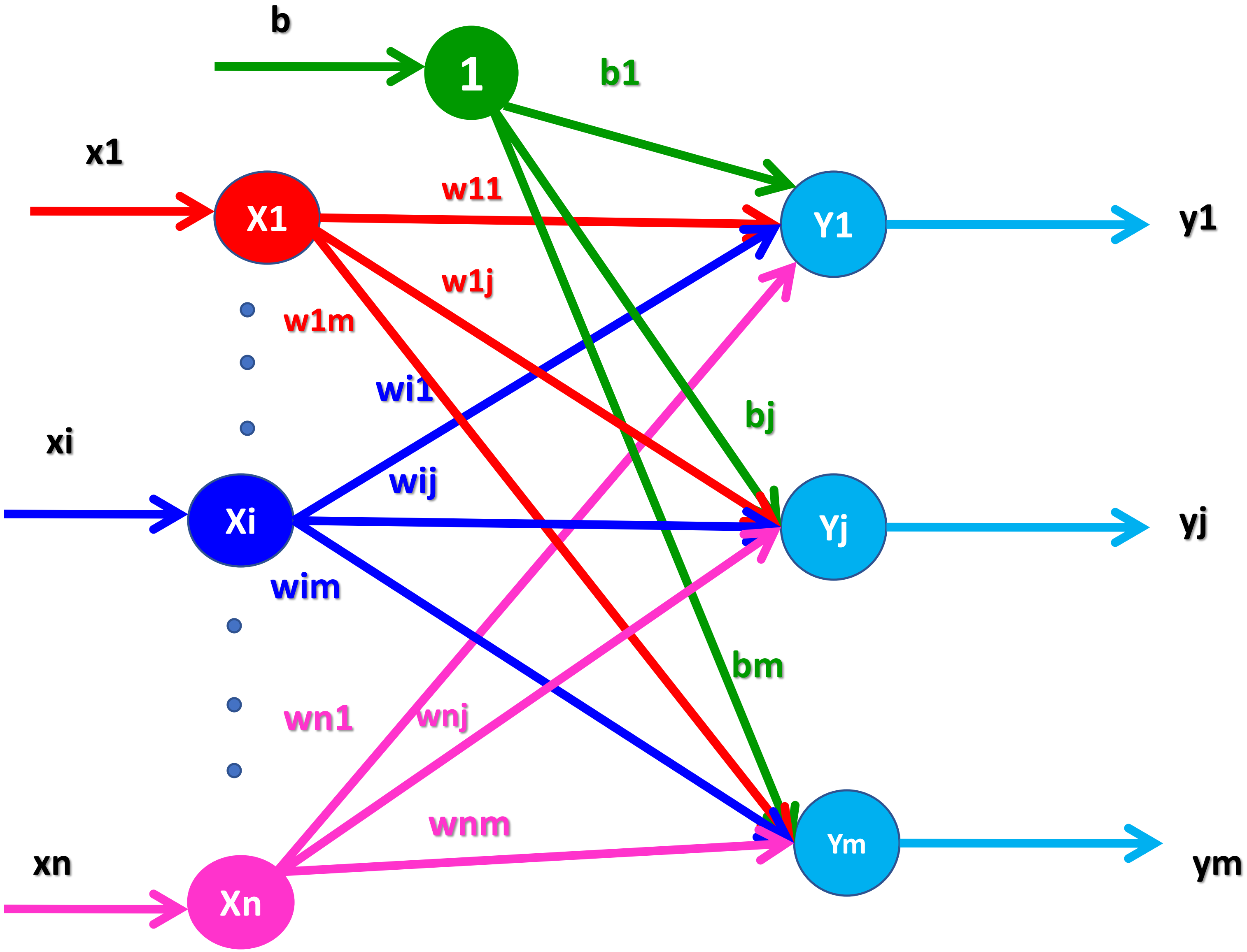
$b(new)=b(old)$

Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met , then start again from step 2

- Notes:
    - Learning occurs only when a sample has y != t
    - Otherwise, it’s similar to Hebb Learning rule
    - Two loops, a completion of the inner loop (each sample is used once) is called an epoch
- **Stop condition**
    - **When no weight is changed in the current epoch, or**
    - **When pre-determined number of epochs is reached**


x0
1
x1
X1
b
w1
wi
xi
Xi
Y
y
wn
xn
Xn

PERCEPTRON TRAINING ALGORITHM FOR MULTIPLE OUTPUT CLASS


b
1
b1
x1
X1
w11
Y1
y1
w1j
w1m
wi1
bj
xi
Xi
wij
Yj
yj
wim
bm
wn1
wnj
wnm
Ym
ym
xn
Xn

Step 0: Initialize the weight and bias (to zero). Also initialize learning rate α ($0 < \alpha <= 1$) For simplicity α set to 1

Step 1: Perform step 2- 6 until the final stopping condition is false

Step 2: Perform step 3-5 for each training pair indicated by s:t

Step 3: Set the activation of each input unit i=1 to n

$$x_i = s_i$$

Step 4: Calculate the output response of each output from j=1 to m . To do so first obtain the net input

$$y_{in\ j} = b_j + \sum_{i=1}^{n} x_i w_{i\ j}$$

'n' is the number of neurons in the layer

Then apply the activation function over net input to get the output

$$y_j = f(y_{in\ j}) = \begin{cases} 1 & \text{If } y_{in} > \theta \\ 0 & \text{If } -\theta <= y_{in} <= \theta \\ -1 & \text{If } y_{in} < -\theta \end{cases}$$

Step 5: Weight and bias adjustment: from j=1 to m and i=1 to n

If $y_j \neq t_j$ then $wi_j(new) = wi_j(old) + \alpha\, t_j\, xi_j$

$b_j(new) = b_j(old) + \alpha t_j$

else $wi_j(new) = wi_j(old)$

$b_j(new) = b_j(old)$

Step 6: Train the network until there is no weight change. This is the stopping condition for the network. If this condition is not met , then start again from step 2

PERCEPTRON NETWORK TESTING ALGORITHM

Step 0: The initial weight used here are taken from the training algorithms (the final weight obtained during training)

Step 1: For each input vector X to be classified perform step 2 – 3

Step 2: Set activation of the input unit

Step 3: Obtain the response of the output unit

$$y_{in} = \sum x_i w_i$$

$$y=f(y_{in}) = \begin{cases} 1 & \text{If } y_{in} > \theta \\ 0 & \text{If } -\theta <= y_{in} <= \theta \\ -1 & \text{If } y_{in} < -\theta \end{cases}$$

✓In case of perceptron network , it can be used for linear seperability concept

✓Here the separating line may be based on the values of threshold

✓The condition for separating the response from region of positive to region of zero is

$$w_1x_1+w_2x_2 +b > \theta$$

✓ The condition for separating the response from region of zero to region of negative is

$$w_1x_1+w_2x_2 +b < - \theta$$

✓The condition above are stated for a single layer perceptron network with two input neuron and one output neuron and one bias

Implement AND function using perceptron network for bipolar input and targets

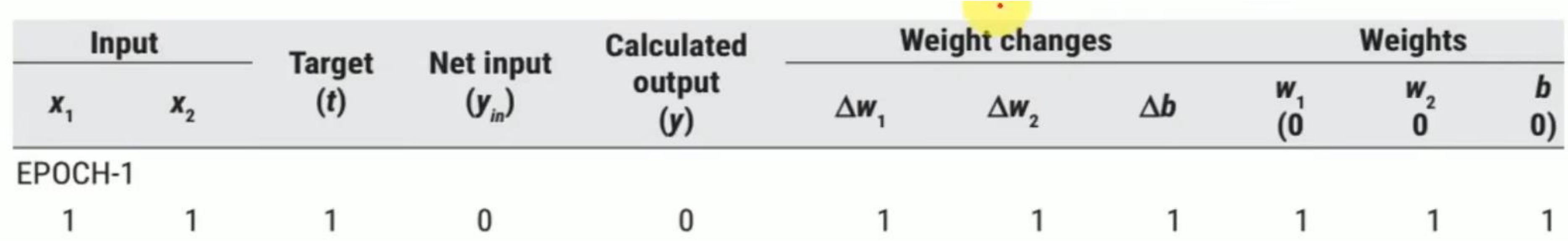
x1	x2	y
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

- X1 = l, X2 = 1 and t = 1, with weights and bias, w1 = 0, W2 = 0 and b=0

- Calculate the net input

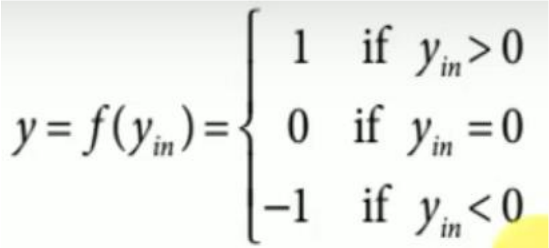
  $$Yin = b+ X_1 W_1+X_2 W_2$$

  =0+1 x 0+1x 0=0



Input		Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1										
1	1	1	0	0	1	1	1	1	1	1

- Yin = b+ X_1 W_1+X_2 W_2
  =0+1 x 0+1x 0=0



$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$

- Y=0
- Is t==y

ΔW1= $\alpha t x_1$

ΔW2= $\alpha t x_2$

Δb= αt

w_i(new)=w_i(old) +$\alpha t x_i$

w_1(new)=0+1x1x1

W1(new)=1

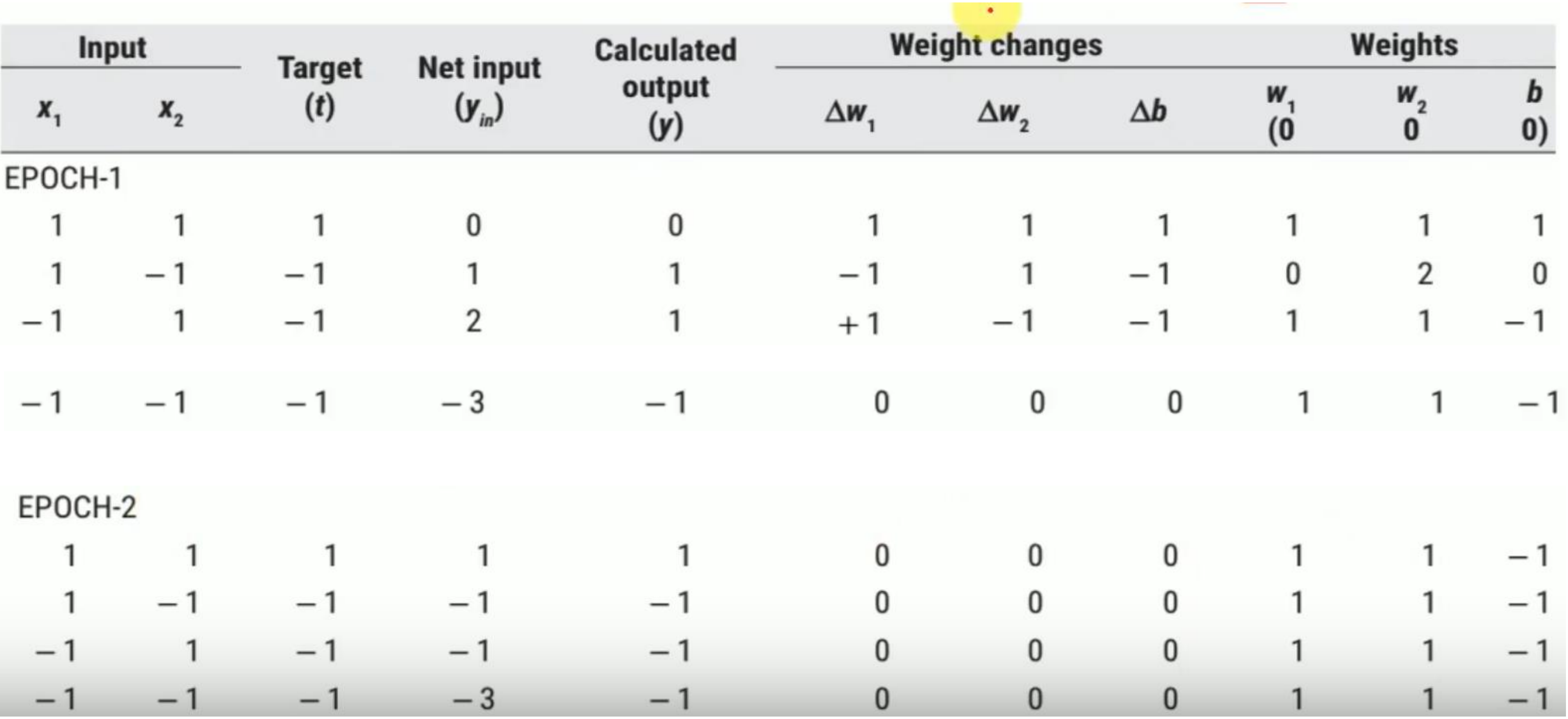
w_2(new)=w_2(old) +$\alpha t x_2$

= 0+1x1x1=1

b(new)=b(old)+αt

0+1x1

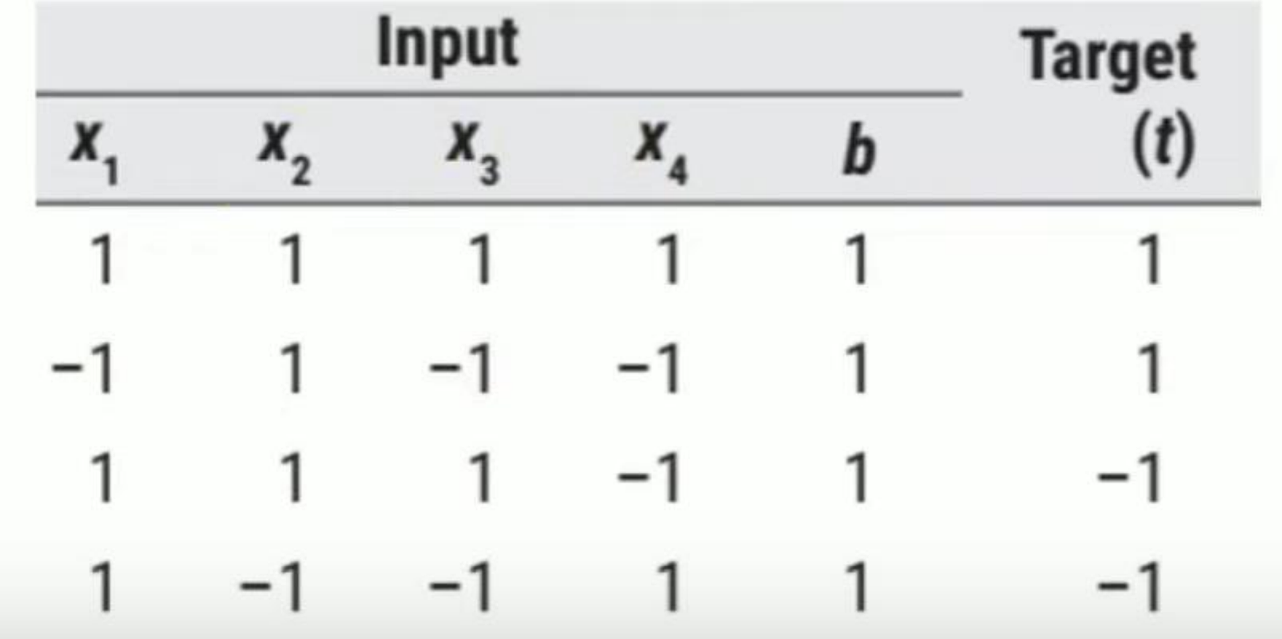
=1



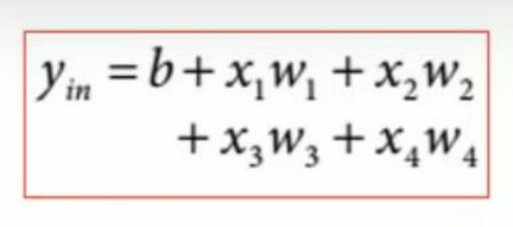
Input		Target	Net input	Calculated output	Weight changes			Weights		
x_1	x_2	(t)	(y_{in})	(y)	Δw_1	Δw_2	Δb	w_1 (0	w_2 0	b 0)
EPOCH-1										
1	1	1	0	0	1	1	1	1	1	1
1	−1	−1	1	1	−1	1	−1	0	2	0
−1	1	−1	2	1	+1	−1	−1	1	1	−1
−1	−1	−1	−3	−1	0	0	0	1	1	−1
EPOCH-2										
1	1	1	1	1	0	0	0	1	1	−1
1	−1	−1	−1	−1	0	0	0	1	1	−1
−1	1	−1	−1	−1	0	0	0	1	1	−1
−1	−1	−1	−3	−1	0	0	0	1	1	−1

- Find the weights required to perform the following classification using perceptron network.
- The vectors (1, 1, 1, 1) and (- 1, 1 - 1, - 1) are belonging to the class 1, vectors (1, 1, 1, - 1) and (1, - 1, - 1, 1) are belonging to the class -1.

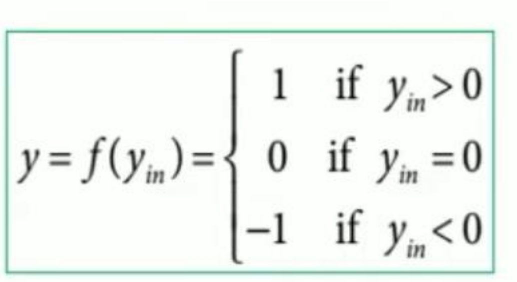
  - Assume learning rate as 1
  - Initial weight =0



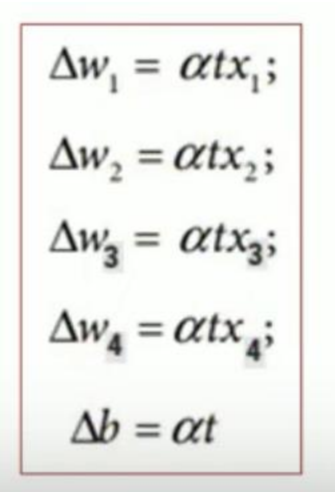
Input					Target
x_1	x_2	x_3	x_4	b	(t)
1	1	1	1	1	1
−1	1	−1	−1	1	1
1	1	1	−1	1	−1
1	−1	−1	1	1	−1



$$y_{in} = b + x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4$$



$$y = f(y_{in}) = \begin{cases} 1 & \text{if } y_{in} > 0 \\ 0 & \text{if } y_{in} = 0 \\ -1 & \text{if } y_{in} < 0 \end{cases}$$



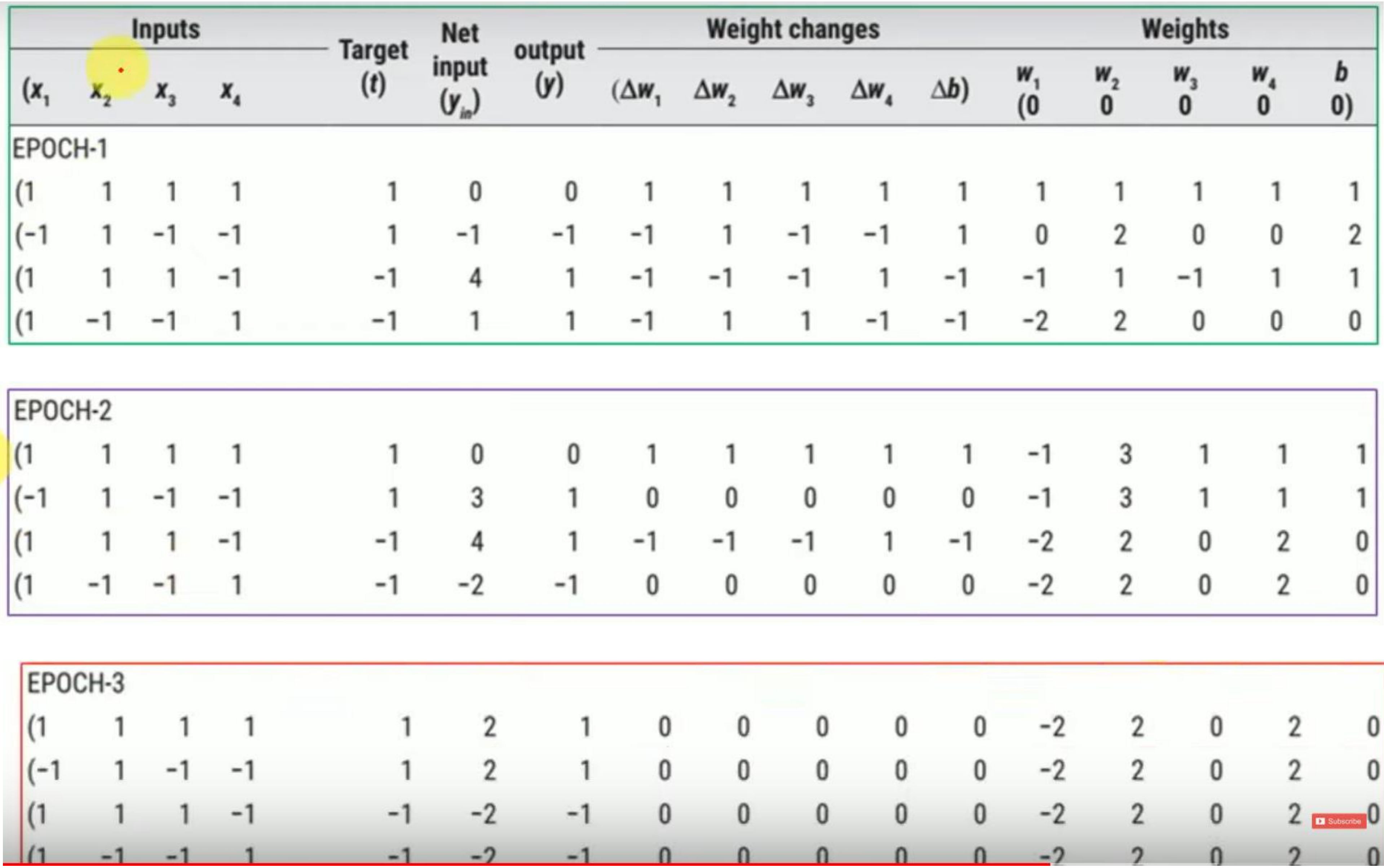
$$\Delta w_1 = \alpha t x_1;$$

$$\Delta w_2 = \alpha t x_2;$$

$$\Delta w_3 = \alpha t x_3;$$

$$\Delta w_4 = \alpha t x_4;$$

$$\Delta b = \alpha t$$



Inputs				Target	Net	output	Weight changes					Weights				
(x_1	x_2	x_3	x_4	(t)	input (y_{in})	(y)	(Δw_1	Δw_2	Δw_3	Δw_4	Δb)	w_1 (0	w_2 0	w_3 0	w_4 0	b 0)
EPOCH-1																
(1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1
(−1	1	−1	−1	1	−1	−1	−1	1	−1	−1	1	0	2	0	0	2
(1	1	1	−1	−1	4	1	−1	−1	−1	1	−1	−1	1	−1	1	1
(1	−1	−1	1	−1	1	1	−1	1	1	−1	−1	−2	2	0	0	0
EPOCH-2																
(1	1	1	1	1	0	0	1	1	1	1	1	−1	3	1	1	1
(−1	1	−1	−1	1	3	1	0	0	0	0	0	−1	3	1	1	1
(1	1	1	−1	−1	4	1	−1	−1	−1	1	−1	−2	2	0	2	0
(1	−1	−1	1	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0
EPOCH-3																
(1	1	1	1	1	2	1	0	0	0	0	0	−2	2	0	2	0
(−1	1	−1	−1	1	2	1	0	0	0	0	0	−2	2	0	2	0
(1	1	1	−1	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0
(1	−1	−1	1	−1	−2	−1	0	0	0	0	0	−2	2	0	2	0

Implement OR function using perceptron network for binary input and bipolar targets up to 3 epochs

Find the weight using perceptron network for ANDNOT function when all the input are presented only one time. Use bipolar input and targets

Find the weight required to perform the following classification using perceptron network. The vectors(1,1,1,1) and (-1,1,-1,-1) are belongs to the class so have the target value 1, vectors (1,1,1,-1) and (1,-1,-1,1) are not belongs to the class so have the target value -1. Assume learning rate 1 and initial weights as 0

Using Perceptron rule, Find the weight required to perform the following classification of the given input patterns shown below

The pattern is shown as 3X3 matrix form in the squares. The “+” symbol represents the value 1 and empty square indicate -1.

Consider “I” belongs to the member of the class so the target value is 1 and “F” does not belongs to the member of the class so the target value is -1

+	+	+
	+	
+	+	+

“I”

+	+	+
+	+	+
+		

“F”

TUTORIAL QUESTIONS

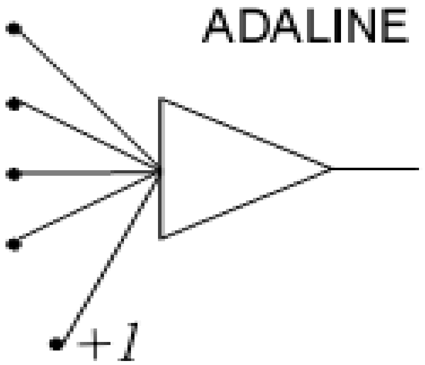
1. **Implement NOR function using perceptron network for bipolar input and targets**
2. **Find the weight required to perform the following classification using perceptron network. The vectors(1,1,-1,-1) and (1,-1,1,-1) are belongs to the class so have the target value 1, vectors (-1,-1,-1,1) and (-1,-1,1,1) are not belongs to the class so have the target value -1. Assume learning rate 1 and initial weights as 0**
3. **Classify the 2D pattern shown in the figure below using perceptron network**

+	+	+
+		
+	+	+

	+	
+	+	+
+	+	+

ADAPTIVE LINEAR NEURON (ADALINE)

In 1959, Bernard Widrow and Marcian Hoff of Stanford developed models they called ADALINE (Adaptive Linear Neuron) and MADALINE (Multilayer ADALINE). These models were named for their use of Multiple ADAptive LINear Elements. MADALINE was the first neural network to be applied to a real world problem. It is an adaptive filter which eliminates echoes on phone lines.



Adaptive Linear Neuron (Adaline)

- The units with **linear activation function** are called linear units.
- A **network with a single linear unit** is called an *Adaline* (adaptive linear neuron).
- That is, in an Adaline, the input-output relationship is linear.
- Adaline *uses* bipolar activation for its **input signals** and its **target output**.
- The **weights** between the input and the output are adjustable.
- The **bias** in Adaline acts like an adjustable weight, whose connection is from a unit with activations being always 1.

Adaptive Linear Neuron (Adaline)

- Adaline is a net which has only **one output unit**.
- The Adaline network **may be trained** using **delta rule.**
- The **delta rule** may also be called as *least mean square* (LMS) rule or Widrow-Hoff rule.
- This **learning rule** is found to minimize the mean squared error between the **activation** and the **target value**.

Delta Rule for Single Output Unit

- The **Widrow-Hoff rule** is very similar to percepton learning rule. However, their origins are different.
- The **perceptron learning rule** originates
  - from the Hebbian assumption
- The **delta rule** is derived
  - from the **gradient- descet method** (it can be generalized to more than one layer) **Gradient descent** is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost). Gradient descent is best used when the parameters cannot be calculated analytically
- Also, the **perceptron learning rule** stops after a finite number of learning steps, but the **gradient-descent** approach continues forever, converging only asymptotically to the solution.

ADALINE LEARNING RULE

Adaline network uses Delta Learning Rule. This rule is also called as Widrow-Hoff Learning Rule or Least Mean Square Rule. The delta rule for adjusting the weights is given as (i = 1 to n):

$$\Delta w_i = \alpha(t - y_{in})x_i$$

$$\begin{aligned}
\Delta w_i &= \text{weight change} \\
\alpha &= \text{learning rate} \\
x &= \text{vector of activation of input unit} \\
y_{in} &= \text{net input to output unit}, i.e., Y = \sum_{i=1}^{n} x_i w_i \\
t &= \text{target output}
\end{aligned}$$

ADALINE LEARNING RULE

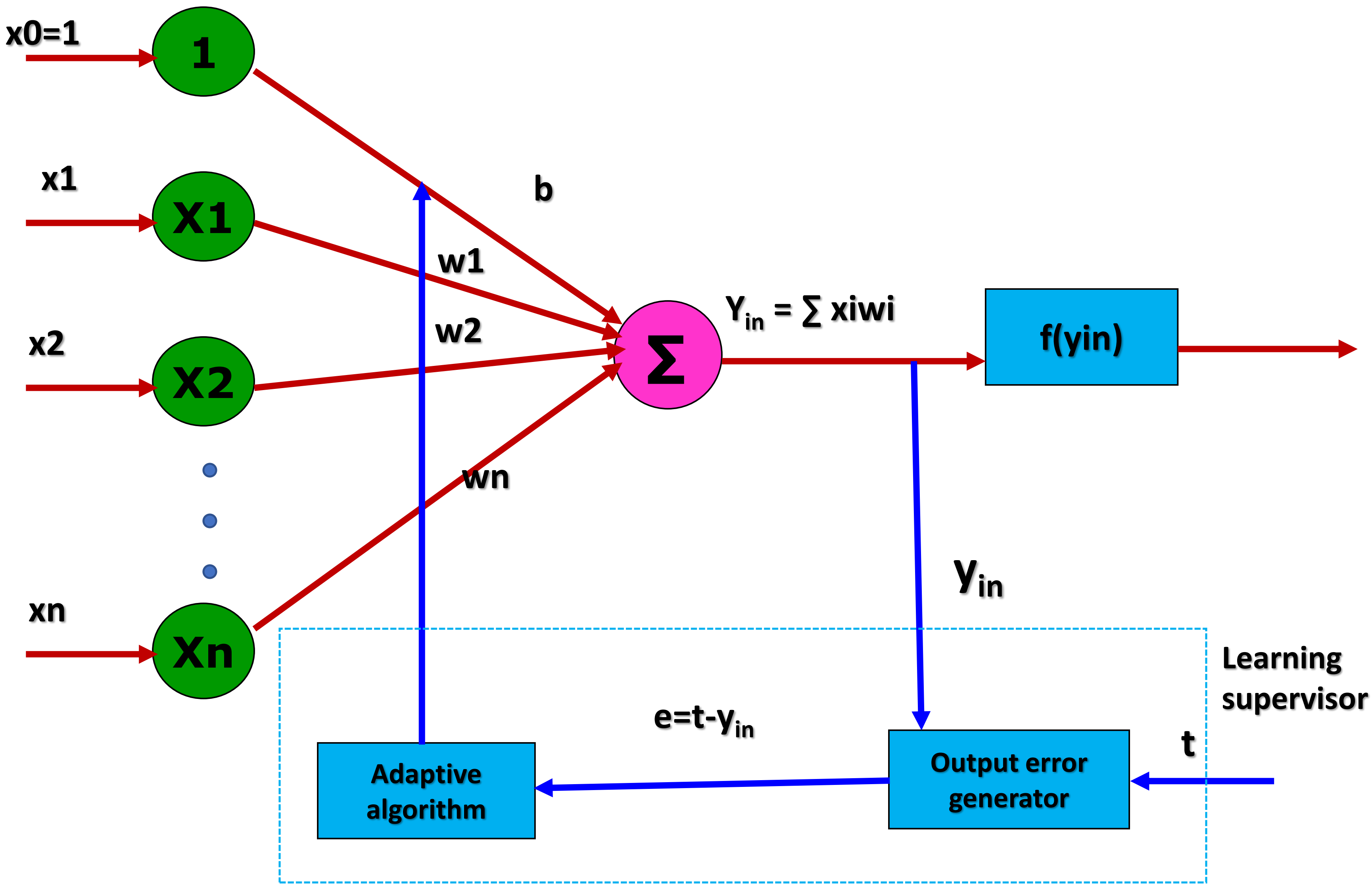
- ✓Similar to Perceptron rule
- ✓Perceptron rule is originated from Hebbian Assumption
- ✓Delta rule is from Gradient Descent method
- ✓ Perceptron rule stops after a finite number of learning steps
- ✓Delta rule continues forever, converging only asymptotically to the solution
- ✓Delta rule update the weight between the connection so as to minimize the difference between the net input to the output unit and the target value
- ✓Major aim is to minimize the error over all training patterns

Perceptron	Delta
Originates from hebbian assumption	Derived from gradient-descent method
Stops after a finite number of learning steps	Continuous forever converging asymptotically to the solution
	Minimizes error over all training patterns

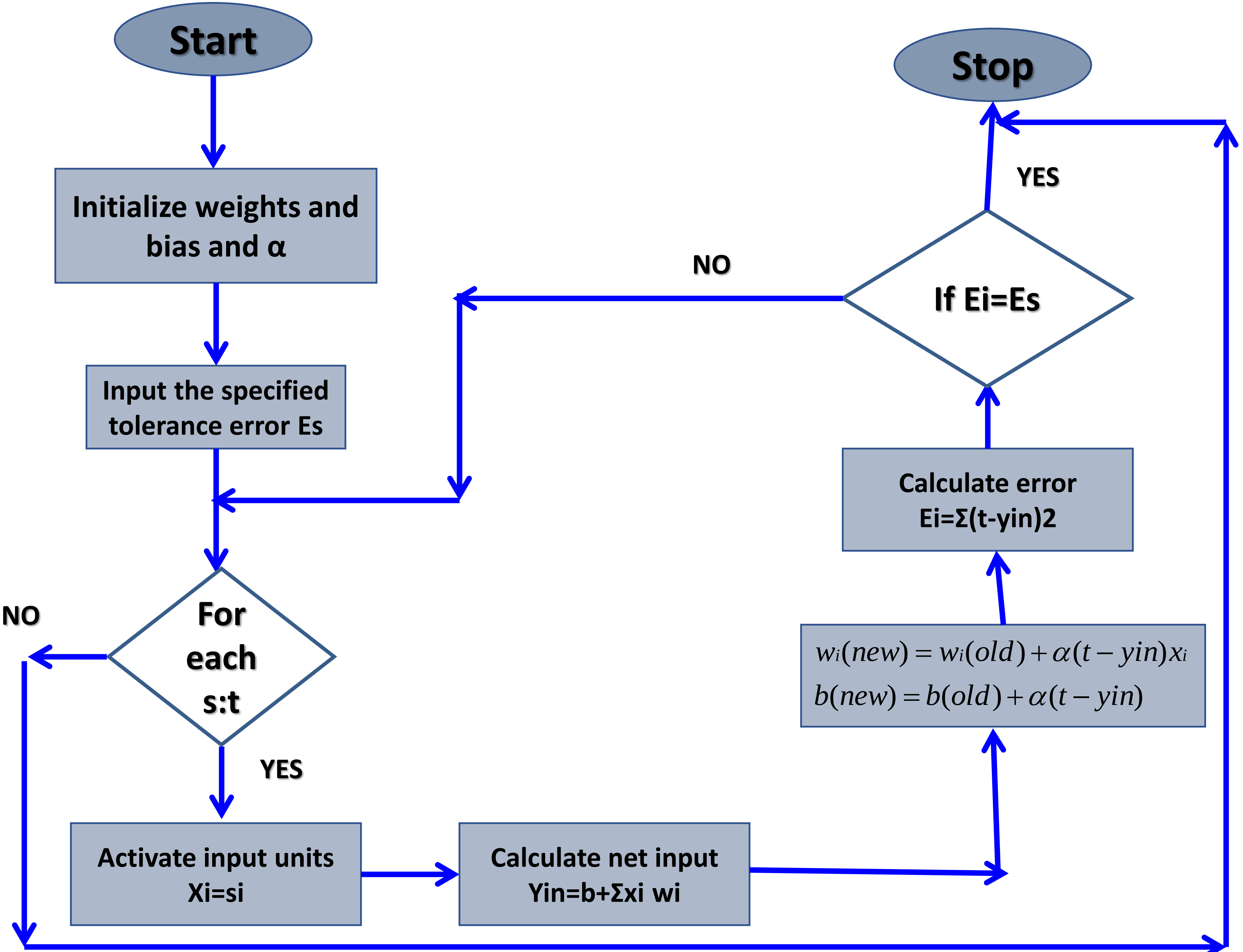
ARCHITECTURE

- ✓Adaline is a single unit neuron, which receives the input from several units and also from units called bias
- ✓Basic Adaline model consist of trainable weights
- ✓Input are either of 2 values(+1 or -1) and the weight have signs(positive or negative)
- ✓Initially random weights are assigned
- ✓The net input is calculated is applied to a quantizer transfer function that restore the output to +1 or -1
- ✓Adaline model compare the actual output with the target output and on the basis of training algorithm, the weights are adjusted

ARCHITECTURE



TRAINING ALGORITHM



Start

Initialize weights and bias and α

Input the specified tolerance error Es

For each s:t

NO

YES

Activate input units Xi=si

Calculate net input Yin=b+Σxi wi

$w_i(new) = w_i(old) + \alpha(t - yin)x_i$

$b(new) = b(old) + \alpha(t - yin)$

Calculate error Ei=Σ(t-yin)2

If Ei=Es

NO

YES

Stop

ADALINE TRAINING ALGORITHM

Step 0: Weights and bias are set to some random values other than zero. Learning rate α is set

Step 1: Perform Steps 2-6 when stopping condition is false.

Step 2: Perform steps 3-5 for each bipolar training pair s:t

Step 3: Set activations for input units i=1 to n xi=si

Step 4: Calculate the net input to the output unit

$$y_{in} = b + \sum_{i=1}^{n} x_i w_i$$

Step 5: Update the weights and bias for i=1 to n

$$wi(new) = wi(old) + \alpha(t - yin)xi$$

$$b(new) = b(old) + \alpha(t - y_{in})$$

Step 6: If highest weight change that occurred during training is smaller than a specified tolerance then stop the training else continue. (Stopping condition)

TESTING ALGORITHM

- Step 0: Initialize the weights(from training algo)
- Step 1: Perform steps2-4 for each bipolar input vector x
- Step 2: Set the activations of the input units to x
- Step 3: Calculate the net input

$$y_{in} = b + \sum x_i w_i$$

- Step 4: Apply the activation function over the net input calculated

$$y = \begin{cases} 1 & \text{If } y_{in} >= 0 \\ -1 & \text{If } y_{in} < \quad 0 \end{cases}$$

Design ADALINE for OR Function

- Initially, all weights are assumed to be small random values, say 0.1, and set learning rule to 0.1.
- Also, set the least squared error to 2.
- The weights will be updated until the total error is greater than the least squared error.

x_1	x_2	t
1	1	1
1	-1	1
-1	1	1
-1	-1	-1

- Calculate the net input

$$y_{in} = b + \sum x_i w_i$$

- Y_{in} = 0.1*1+0.1*1+.1 = 0.3

- Now compute, $(t-y_{in})=(1-0.3)=0.7$
- Now, update the weights and bias

$$w_i(new) = w_i(old) + (t - y_{in})x_i$$
$$w_1(new) = 0.1 + 0.1(1 - 0.3)1 = 0.17$$
$$w_2(new) = 0.1 + 0.1(1 - 0.3)1 = 0.17$$

$$b(new) = b(old) + (t - y_{in})$$
$$b(new) = 0.1 + 0.1(1 - 0.3) = 0.17$$

- calculate the error

$$error = (t - y_{in})^2 = 0.7^2 = 0.49$$

- Similarly, repeat the same steps for other input vectors and you will get.

x_1	x_2	t	y_{in}	$(t-y_{in})$	Δw_1	Δw_2	Δb	w_1 (0.1)	w_2 (0.1)	b (0.1)	$(t-y_{in})$^2
1	1	1	0.3	0.7	0.07	0.07	0.07	0.17	0.17	0.17	0.49
1	-1	1	0.17	0.83	0.083	-0.083	0.083	0.253	0.087	0.253	0.69
-1	1	1	0.087	0.913	-0.0913	0.0913	0.0913	0.1617	0.1783	0.3443	0.83
-1	-1	-1	0.0043	-1.0043	0.1004	0.1004	-0.1004	0.2621	0.2787	0.2439	1.01

TUTORIAL QUESTIONS

1. Use ADALINE network to train ANDNOT function with bipolar input and targets. Perform 2 epoch of training
2. Implement AND function using ADALINE
3. Using Delta rule find the weight required to perform following classifications: Vectors(1,1,-1,-1) and (-1,-1,-1,-1) belongs to the class having target value 1: vectors (1,1,1,1) and (-1,-1,1,-1) re not belong to the class having target value -1.Use learning rate of 0.5 and assume random values weight. Also test the network

Back-Propagation Network

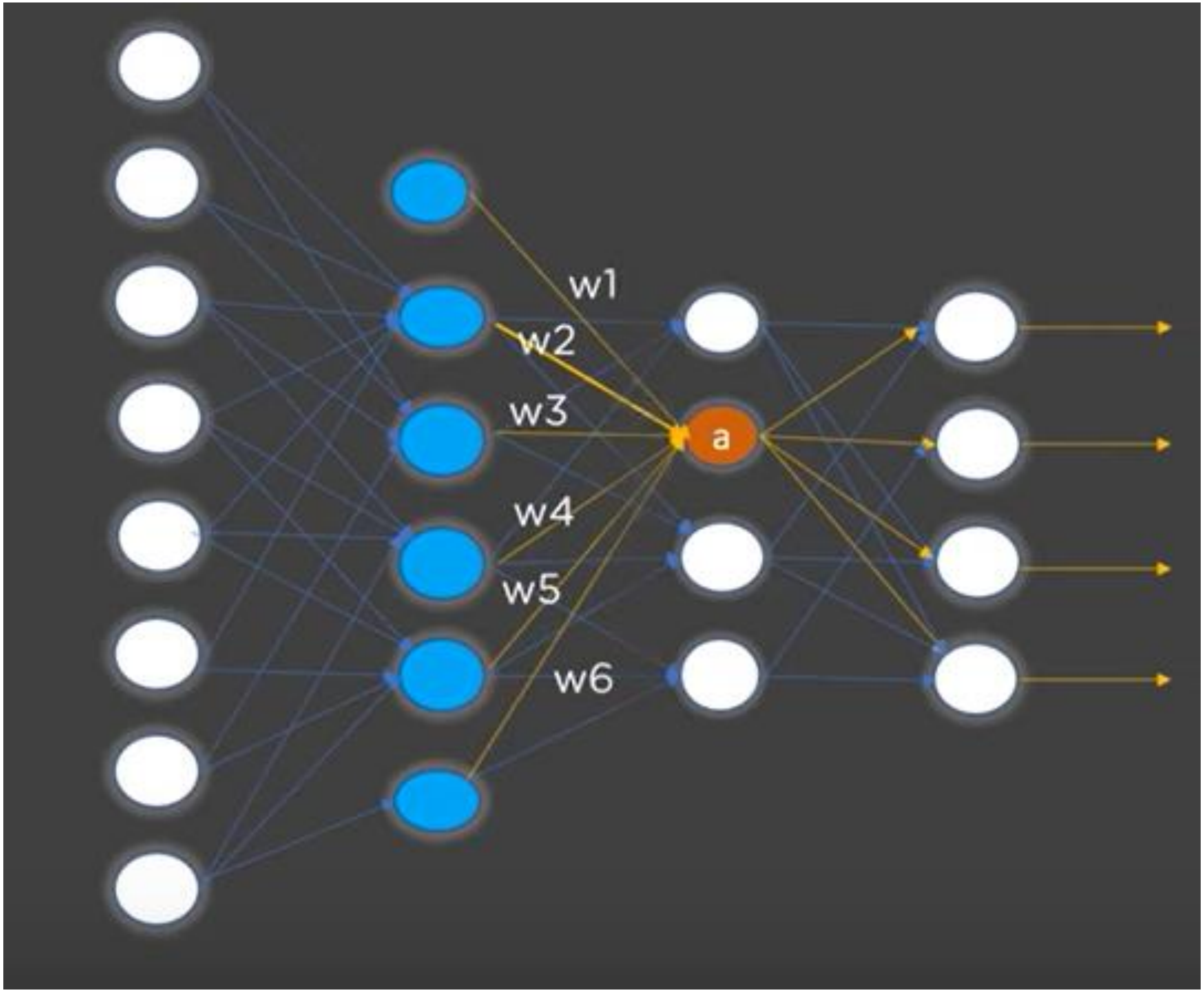
- The back-propagation learning algorithm is one of the most important developments in **neural networks** (Bryson and Ho, 1969; Werbos, 1974; Lecun, 1985; Parker, 1985; Rumelhan, 1986).
- This **network** has reawakened the **scientific** and **engineering community** to the modeling and processing of numerous phenomena using neural networks.
- This learning algorithm is applied to **multilayer feed-forward network** consisting on processing elements with continuous differentiable activation functions.
- The networks associated with **back-propagation learning algorithm** are al*so called* ***back-propagation network*** **(BPNs).**

Back-Propagation Network

- For a given set of training input-output pair, this **algorithm** provides a procedure for changing the weights in a BPN to classify the given input patterns correctly.
- The basic concept for this weight update algorithm is simply the **gradient-descent method** as used in the case of simple perceptron networks with differentiable units.
- This is a method where the error is propagated back to the hidden unit.
- The **aim** of the neural network is
  - ➢ *to* train the net to achieve a balance between the **net's ability to respond (memorization)** and its ability **to give reasonable responses to the input** that is **similar** but **not identical** to the one that is used in training (generalization).

Back-Propagation Network

- The back-propagation algorithm is different from other networks in respect to the process by which the weights are calculated during the learning period of the network.
- The **general difficulty** with the multilayer perceptron is calculating the weights of the hidden layers in an efficient way that would result in a very small or zero output error.
- When the **hidden layers are increased** the network training becomes more complex.



w1
w2
w3
w4
w5
w6
a

Back-Propagation Network

- To **update weights**, the error must be calculated. The error, Which is the difference between the actual (calculated) and the desired (target) output, is easily measured at the output layer.
- It should be noted that at the **hidden layers**, there is no direct information of the error.
- Therefore, other techniques should be used to calculate an error at the hidden layer, **which will cause minimization of the output error**, and this is the ultimate goal.

Back-Propagation Network

- The training of the BPN is done in three stages –
  - the feed-forward of the input training pattern,
  - the calculation and back-propagation of the error,
  - updation of weights.

Back-Propagation Network-Architecture

- A back-propagation neural network *is* a **multilayer, feed forward neural network** consisting of an input layer, a hidden layer and an output layer.
- The neurons present in the hidden and output layers have biases, which are the connections from the units whose activation is always 1.
- The bias terms also acts as weights.
- Figure shows the architecture of a BPN, depicting only the direction of information flow for the feed forward phase.

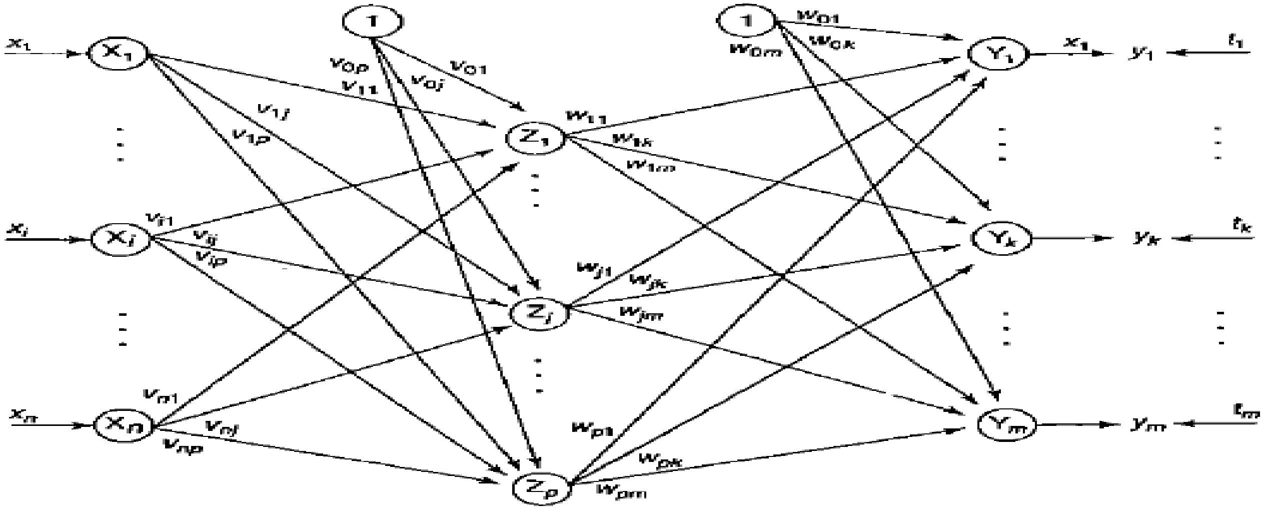


Figure Architecture of a back-propagation network.

Back-Propagation Network-Architecture

- During the back propagation phase of learning , *sig*nals are sent in the reverse direction
- The inputs sent to the BPN and the output obtained from the net could be e1ther binary (0, 1) or bipolar (-1, + 1).
- The activation function could be any function which increases monotonically and is also differentiable.

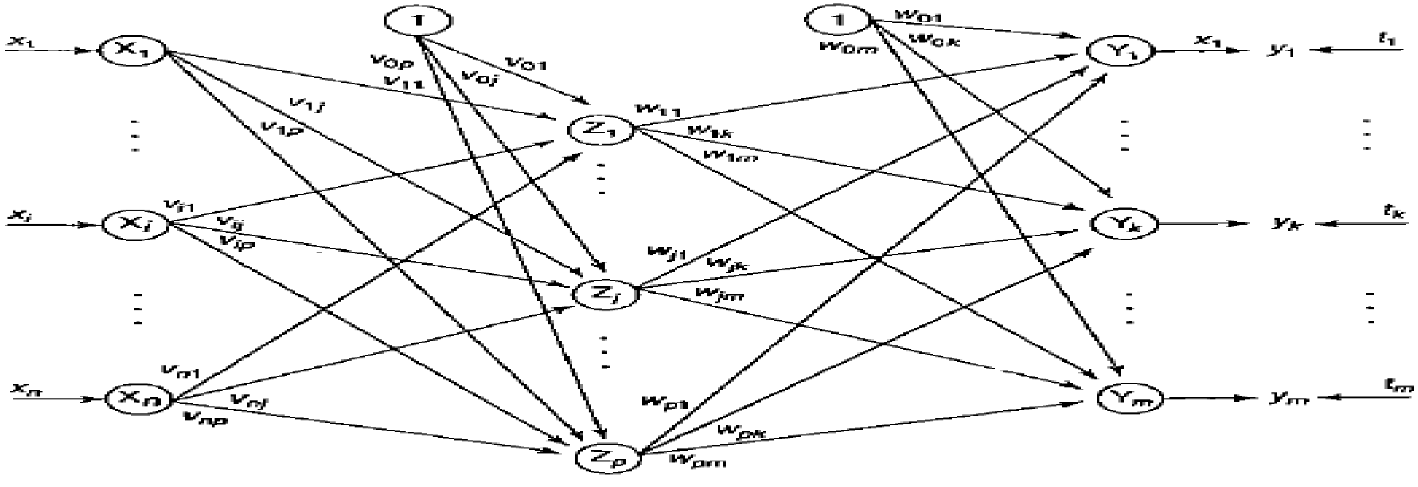


Figure Architecture of a back-propagation network.

Back-Propagation Network-Architecture

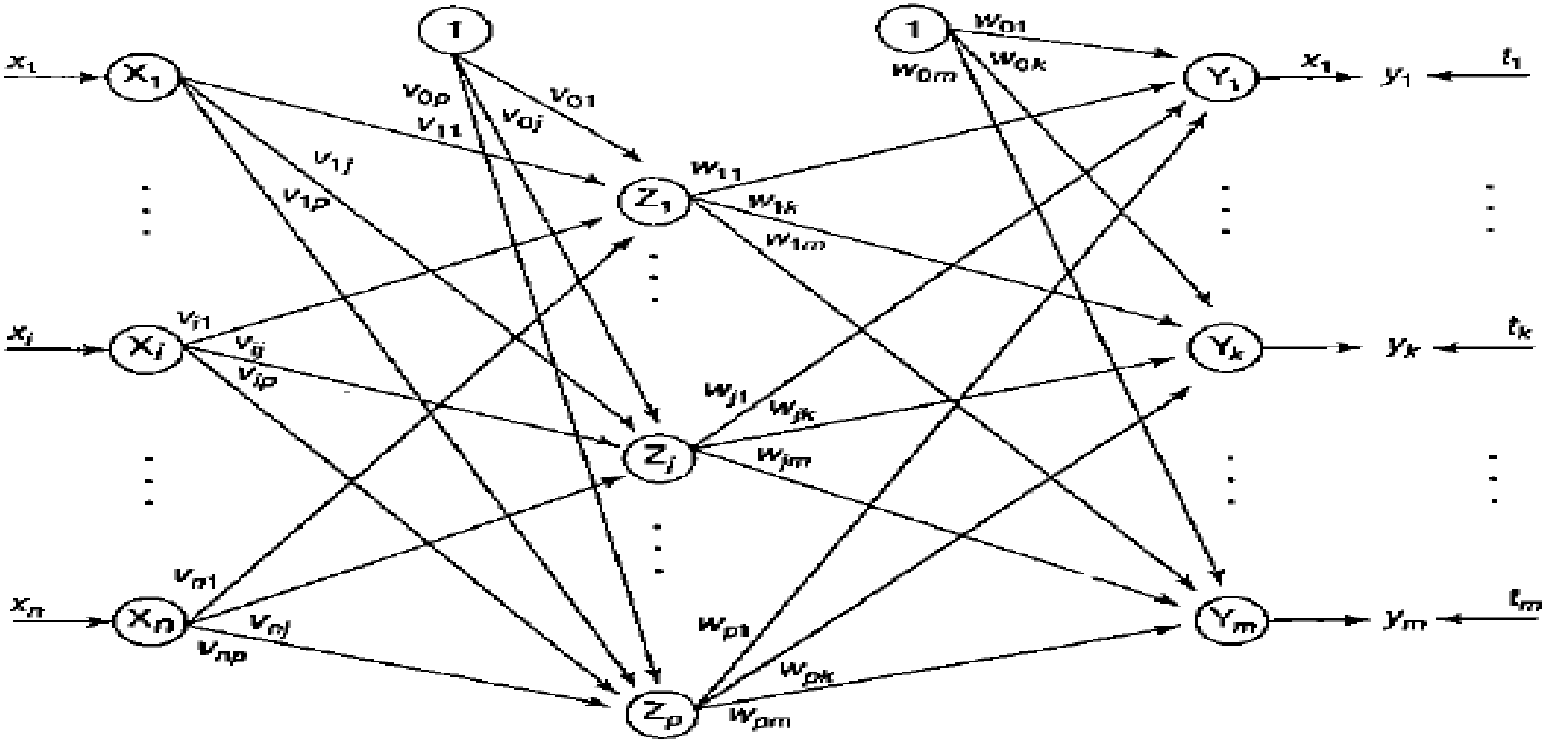


Figure Architecture of a back-propagation network.

x = input training vector $(x_1, \ldots, x_i, \ldots, x_n)$

t = target output vector $(t_1, \ldots, t_k, \ldots, t_m)$

α = learning rate parameter

x_i = input unit i. (Since the input layer uses identity activation function, the input and output signals here are same.)

v_{0j} = bias on jth hidden unit

w_{0k} = bias on kth output unit

z_j = hidden unit j. The net input to z_j is

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

and the output is

$$z_j = f(z_{inj})$$

y_k = output unit k. The net input to y_k is

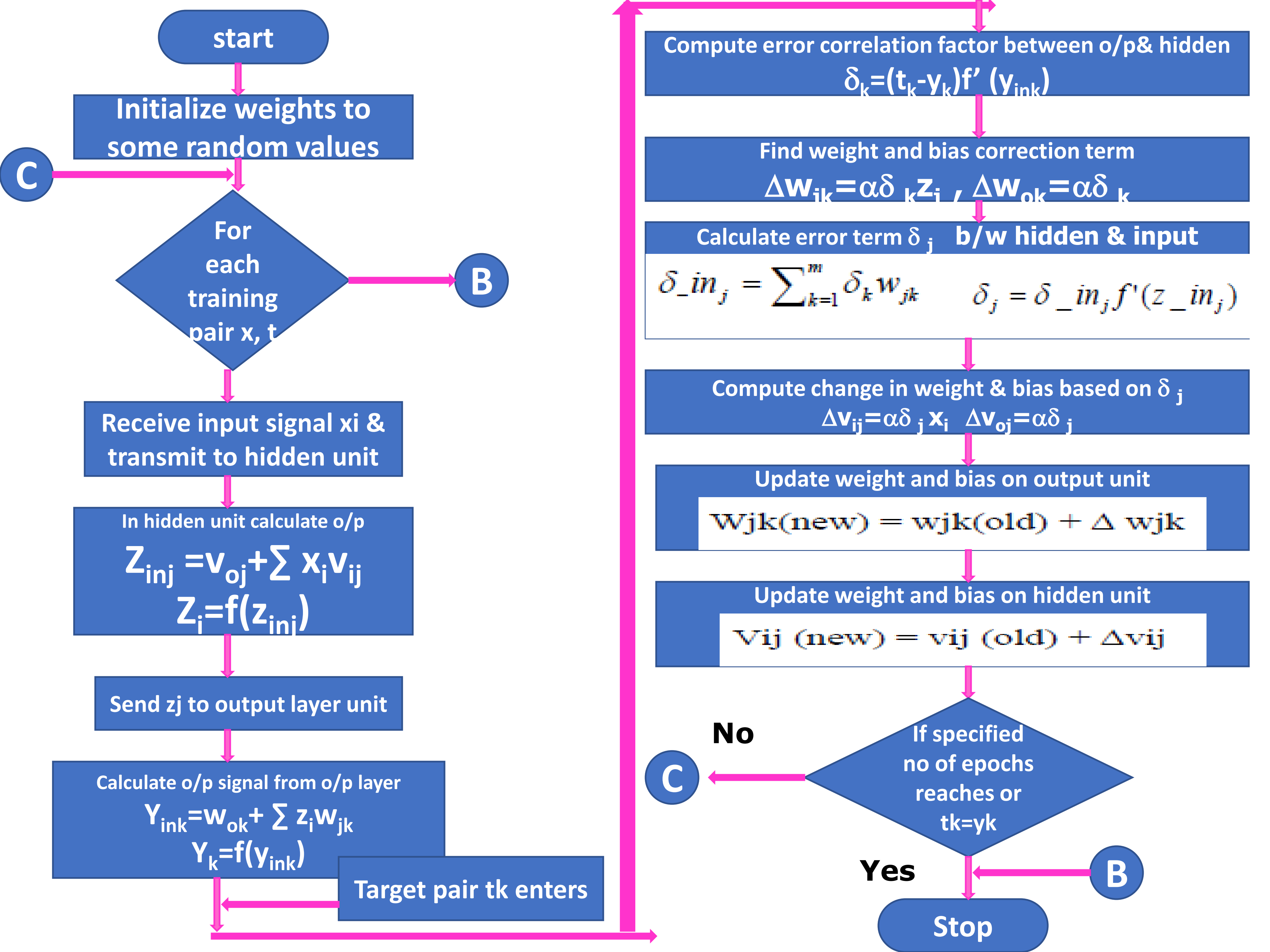
$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and the output is

$$y_k = f(y_{ink})$$

δ_k = error correction weight adjustment for w_{jk} that is due to an error at output unit y_k, which is back-propagated to the hidden units that feed into unit y_k

δ_j = error correction weight adjustment for v_{ij} that is due to the back-propagation of error to the hidden unit z_j.



start

Initialize weights to some random values

C

For each training pair x, t

B

Receive input signal xi & transmit to hidden unit

In hidden unit calculate o/p

$Z_{inj} = v_{oj} + \sum x_i v_{ij}$

$Z_i = f(z_{inj})$

Send zj to output layer unit

Calculate o/p signal from o/p layer

$Y_{ink} = w_{ok} + \sum z_i w_{jk}$

$Y_k = f(y_{ink})$

Target pair tk enters

Compute error correlation factor between o/p& hidden

$\delta_k = (t_k - y_k) f'(y_{ink})$

Find weight and bias correction term

$\Delta w_{jk} = \alpha \delta_k z_j$, $\Delta w_{ok} = \alpha \delta_k$

Calculate error term δ_j b/w hidden & input

$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$ $\delta_j = \delta\_in_j f'(z\_in_j)$

Compute change in weight & bias based on δ_j

$\Delta v_{ij} = \alpha \delta_j x_i$ $\Delta v_{oj} = \alpha \delta_j$

Update weight and bias on output unit

$Wjk(new) = wjk(old) + \Delta wjk$

Update weight and bias on hidden unit

$Vij\ (new) = vij\ (old) + \Delta vij$

No

C

If specified no of epochs reaches or tk=yk

Yes

B

Stop

Training Algorithm

- Step 0: Initialize weights and learning rate (take some small random values).
- Step 1: Perform Steps 2-9 when stopping condition is false.
- Step 2: Perform Steps 3-8 for each training pair.

Feed-forward phase (Phase I)

Step 3: Each input unit receives input signal x_i and sends it to the hidden unit ($i = 1$ to n).

Step 4: Each hidden unit $z_j (j = 1$ to $p)$ sums its weighted input signals to calculate net input:

$$z_{inj} = v_{0j} + \sum_{i=1}^{n} x_i v_{ij}$$

Calculate output of the hidden unit by applying its activation functions over z_{inj} (binary or bipola sigmoidal activation function):

$$z_j = f(z_{inj})$$

and send the output signal from the hidden unit to the input of output layer units.

Step 5: For each output unit y_k ($k = 1$ to m), calculate the net input:

$$y_{ink} = w_{0k} + \sum_{j=1}^{p} z_j w_{jk}$$

and apply the activation function to compute output signal

$$y_k = f(y_{ink})$$

Back-propagation of error (Phase II)·

Step 6: Each output unit $y_k(k = 1$ to $m)$ receives a target pattern corresponding to the input training pattern and computes the error correction term:

$$\delta_k = (t_k - y_k)f'(y_{ink})$$

The derivative $f'(y_{ink})$ can be calculated as in Section 2.3.3. On the basis of the calculated error correction term, update the change in weights and bias:

$$\Delta w_{jk} = \alpha \delta_k z_j; \quad \Delta w_{0k} = \alpha \delta_k$$

Also, send δ_k to the hidden layer backwards.

Step 7: Each hidden unit $(z_j, j = 1$ to $p)$ sums its delta inputs from the output units:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

The term δ_{inj} gets multiplied with the derivative of $f(z_{inj})$ to calculate the error term:

$$\delta_j = \delta_{inj} f'(z_{inj})$$

The derivative $f'(z_{inj})$ can be calculated depending on whether binary or bipolar sigmoidal function is used. On the basis of the calculated δ_j, update the change in weights and bias:

$$\Delta v_{ij} = \alpha \delta_j x_i; \quad \Delta v_{0j} = \alpha \delta_j$$

Weight and bias updation (Phase III):

Step 8: Each output unit (y_k, $k = 1$ to m) updates the bias and weights:

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \Delta w_{jk}$$

$$w_{0k}(\text{new}) = w_{0k}(\text{old}) + \Delta w_{0k}$$

Each hidden unit (z_j, $j = 1$ to p) updates its bias and weights:

$$v_{ij}(\text{new}) = v_{ij}(\text{old}) + \Delta v_{ij}$$

$$v_{0j}(\text{new}) = v_{0j}(\text{old}) + \Delta v_{0j}$$

Step 9: Check for the stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equals the target output.

Back-Propagation Network-Architecture

Weight and bias adjustment

- Each **output unit** (yk, *k* = 1 to m) updates the bias and weights:

$wjk(new) = wjk(old)+\Delta wjk$ → $wjk(new) = wjk(old)+\alpha\delta kzj$

$w0k(new) = w0k(old)+\Delta w0k$ → $w0k(new) = w0k(old)+\alpha\delta k$

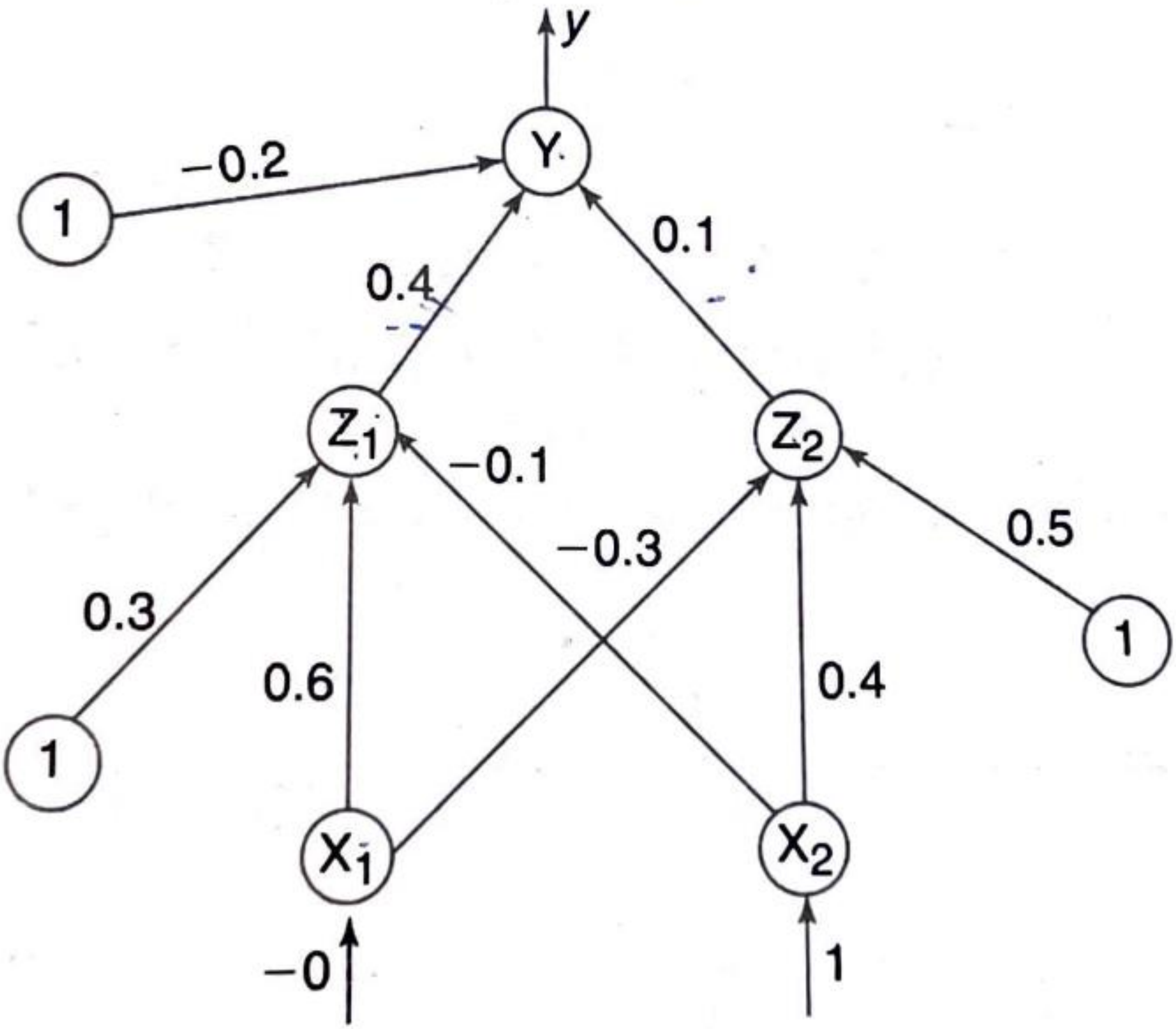
$$\boxed{\Delta wjk = \alpha\delta kzj;\ \Delta wok = \alpha\delta k;}$$

- Each **hidden unit** *(zj, j* = 1 to p) updates its bias and weights:

$vij(new) = vij(old)+\Delta vij$ → $vij(new) = vij(old)+\alpha\delta jxi$

$v0j(new) = v0j(old)+\Delta v0j$ → $v0j(new) = v0j(old)+\alpha\delta j$

$$\boxed{\Delta vij = \alpha\delta jxi;\quad \Delta voj = \alpha\delta j;}$$


y
Y
−0.2
1
0.4
0.1
Z₁
Z₂
−0.1
−0.3
0.5
0.3
0.6
0.4
1
1
X₁
X₂
−0
1

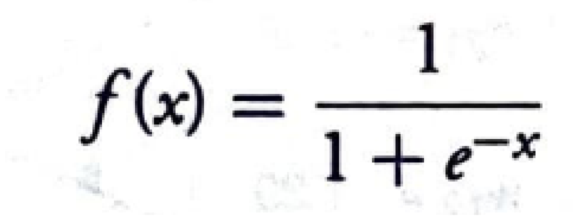
- Calculate the net input: For z_1 layer

$$\begin{aligned} z_{in1} &= v_{01} + x_1 v_{11} + x_2 v_{21} \\ &= 0.3 + 0 \times 0.6 + 1 \times -0.1 = 0.2 \end{aligned}$$

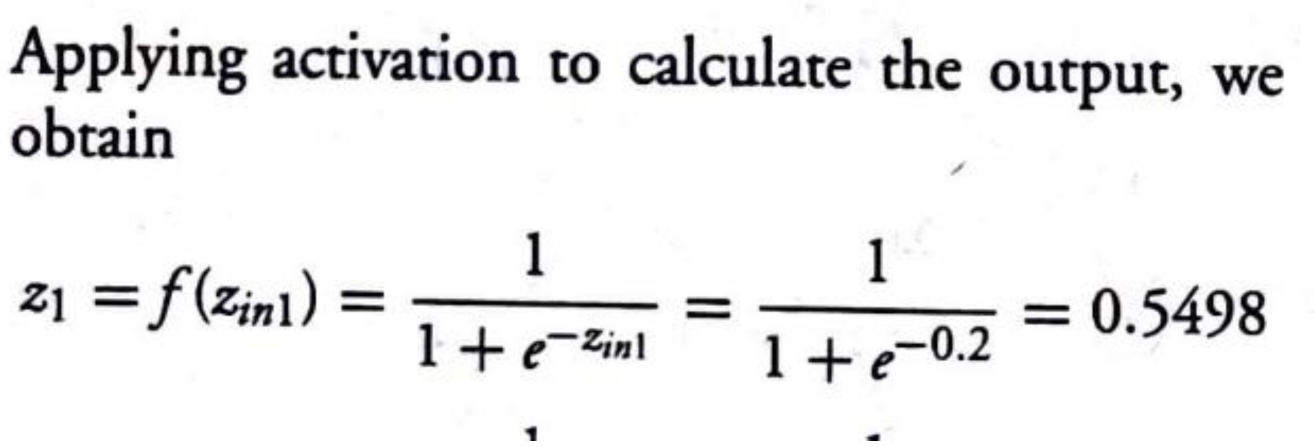
For z_2 layer

$$\begin{aligned} z_{in2} &= v_{02} + x_1 v_{12} + x_2 v_{22} \\ &= 0.5 + 0 \times -0.3 + 1 \times 0.4 = 0.9 \end{aligned}$$

- Activation function used is binary sigmoidal activation

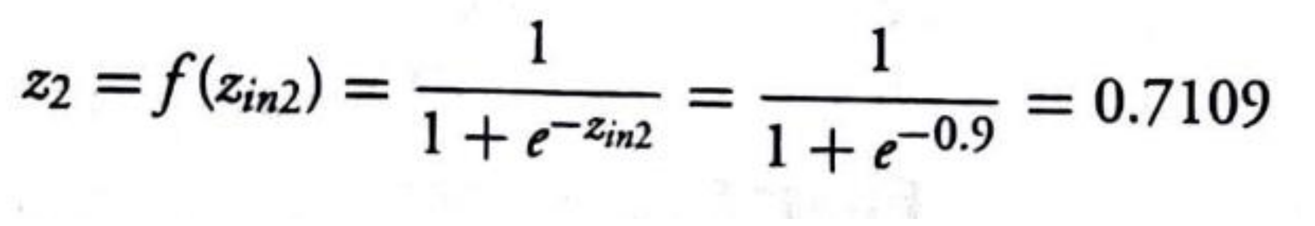


$$f(x) = \frac{1}{1+e^{-x}}$$



Applying activation to calculate the output, we obtain

$$z_1 = f(z_{in1}) = \frac{1}{1+e^{-z_{in1}}} = \frac{1}{1+e^{-0.2}} = 0.5498$$



$$z_2 = f(z_{in2}) = \frac{1}{1+e^{-z_{in2}}} = \frac{1}{1+e^{-0.9}} = 0.7109$$

Calculate the net input entering the output layer.
For y layer

$$\begin{aligned} y_{in} &= w_0 + z_1 w_1 + z_2 w_2 \\ &= -0.2 + 0.5498 \times 0.4 + 0.7109 \times 0.1 \\ &= 0.09101 \end{aligned}$$

Applying activations to calculate the output, we obtain

$$y = f(y_{in}) = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-0.09101}} = 0.5227$$

Compute the error portion δ_k:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

$$f'(y_{in}) = f(y_{in})[1 - f(y_{in})] = 0.5227[1 - 0.5227]$$
$$f'(y_{in}) = 0.2495$$

$$\delta_1 = (1 - 0.5227)\,(0.2495) = \underline{0.1191}$$

Find the changes in weights between hidden and output layer:

$$\begin{aligned}\Delta w_1 &= \alpha\delta_1\, z_1 = 0.25 \times 0.1191 \times 0.5498\\ &= 0.0164\end{aligned}$$

$$\begin{aligned}\Delta w_2 &= \alpha\delta_1\, z_2 = 0.25 \times 0.1191 \times 0.7109\\ &= 0.02117\end{aligned}$$

$$\Delta w_0 = \alpha\delta_1 = 0.25 \times 0.1191 = 0.02978$$

Compute the error portion δ_j between input and hidden layer ($j = 1$ to 2):

$$\delta_j = \delta_{inj} f'(z_{inj})$$

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k w_{jk}$$

$$\delta_{inj} = \delta_1 w_{j1} \quad [\because \text{only one output neuron}]$$

$$\Rightarrow \delta_{in1} = \delta_1 w_{11} = 0.1191 \times 0.4 = 0.04764$$

$$\Rightarrow \delta_{in2} = \delta_1 w_{21} = 0.1191 \times 0.1 = 0.01191$$

Error, $\delta_1 = \delta_{in1} f'(z_{in1})$

$$f'(z_{in1}) = f(z_{in1})[1 - f(z_{in1})]$$

$$= 0.5498[1 - 0.5498] = 0.2475$$

$$\delta_1 = \delta_{in1} f'(z_{in1})$$

$$= 0.04764 \times 0.2475 = 0.0118$$

Error, $\delta_2 = \delta_{in2} f'(z_{in2})$

$$f'(z_{in2}) = f(z_{in2})[1 - f(z_{in2})]$$

$$= 0.7109[1 - 0.7109] = 0.2055$$

$$\delta_2 = \delta_{in2} f'(z_{in2})$$

$$= 0.01191 \times 0.2055 = 0.00245$$

$$\Delta v_{11} = \alpha \delta_1 x_1 = 0.25 \times 0.0118 \times 0 = 0$$

$$\Delta v_{21} = \alpha \delta_1 x_2 = 0.25 \times 0.0118 \times 1 = 0.00295$$

$$\Delta v_{01} = \alpha \delta_1 = 0.25 \times 0.0118 = 0.00295$$

$$\Delta v_{12} = \alpha \delta_2 x_1 = 0.25 \times 0.00245 \times 0 = 0$$

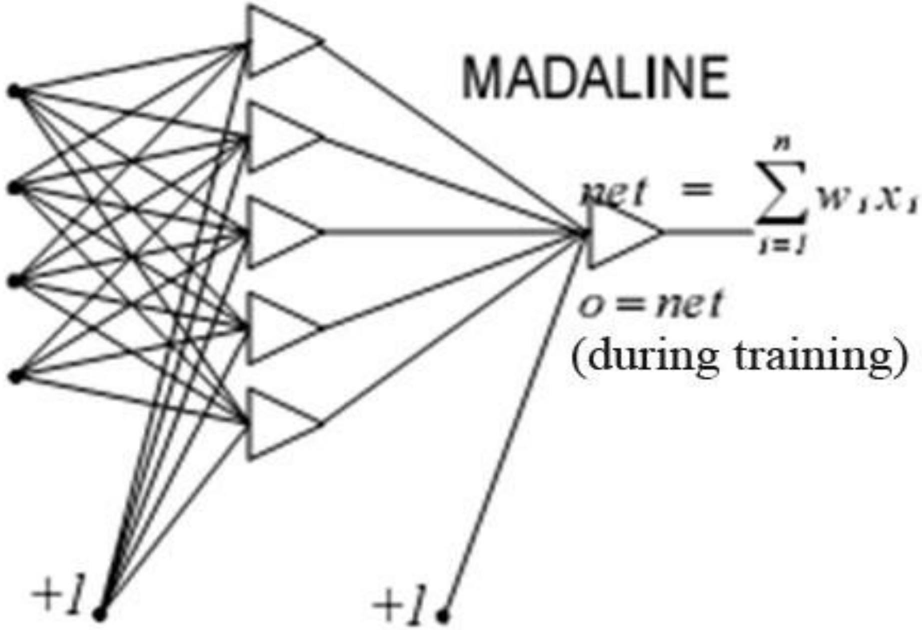
$$\Delta v_{22} = \alpha \delta_2 x_2 = 0.25 \times 0.00245 \times 1 = 0.0006125$$

$$\Delta v_{02} = \alpha \delta_2 = 0.25 \times 0.00245 = 0.0006125$$

$$
\begin{aligned}
v_{11}(\text{new}) &= v_{11}(\text{old}) + \Delta v_{11} = 0.6 + 0 = 0.6 \\
v_{12}(\text{new}) &= v_{12}(\text{old}) + \Delta v_{12} = -0.3 + 0 = -0.3 \\
v_{21}(\text{new}) &= v_{21}(\text{old}) + \Delta v_{21} \\
&= -0.1 + 0.00295 = -0.09705 \\
v_{22}(\text{new}) &= v_{22}(\text{old}) + \Delta v_{22} \\
&= 0.4 + 0.0006125 = 0.4006125 \\
w_{1}(\text{new}) &= w_{1}(\text{old}) + \Delta w_{1} = 0.4 + 0.0164 \\
&= 0.4164 \\
w_{2}(\text{new}) &= w_{2}(\text{old}) + \Delta w_{2} = 0.1 + 0.02117 \\
&= 0.12117 \\
v_{01}(\text{new}) &= v_{01}(\text{old}) + \Delta v_{01} = 0.3 + 0.00295 \\
&= 0.30295 \\
v_{02}(\text{new}) &= v_{02}(\text{old}) + \Delta v_{02} \\
&= 0.5 + 0.0006125 = 0.5006125 \\
w_{0}(\text{new}) &= w_{0}(\text{old}) + \Delta w_{0} = -0.2 + 0.02978 \\
&= -0.17022
\end{aligned}
$$

MADALINE NETWORK

MADALINE is a Multilayer Adaptive Linear Element. MADALINE was the first neural network to be applied to a real world problem. It is used in several adaptive filtering process.

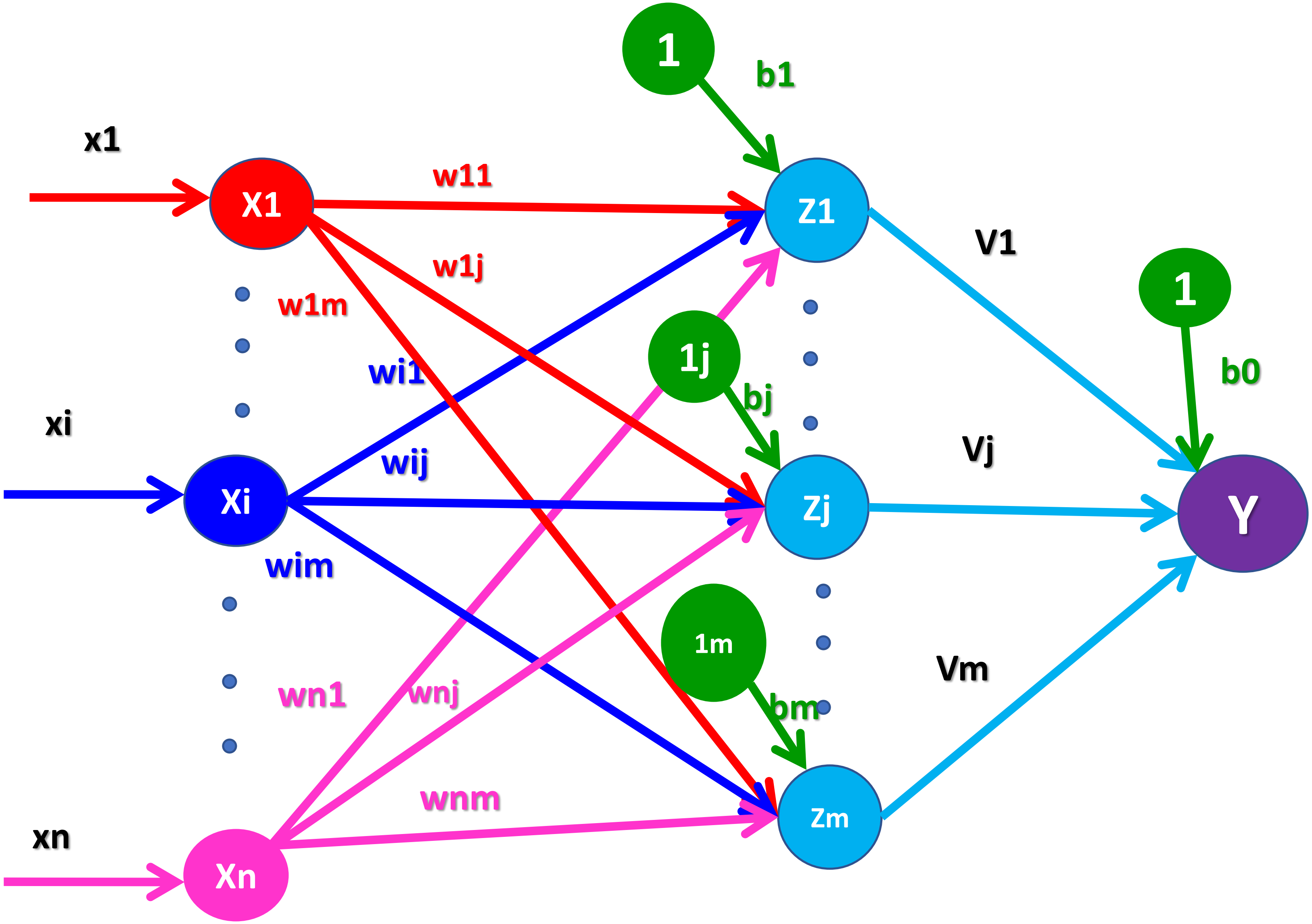


MADALINE

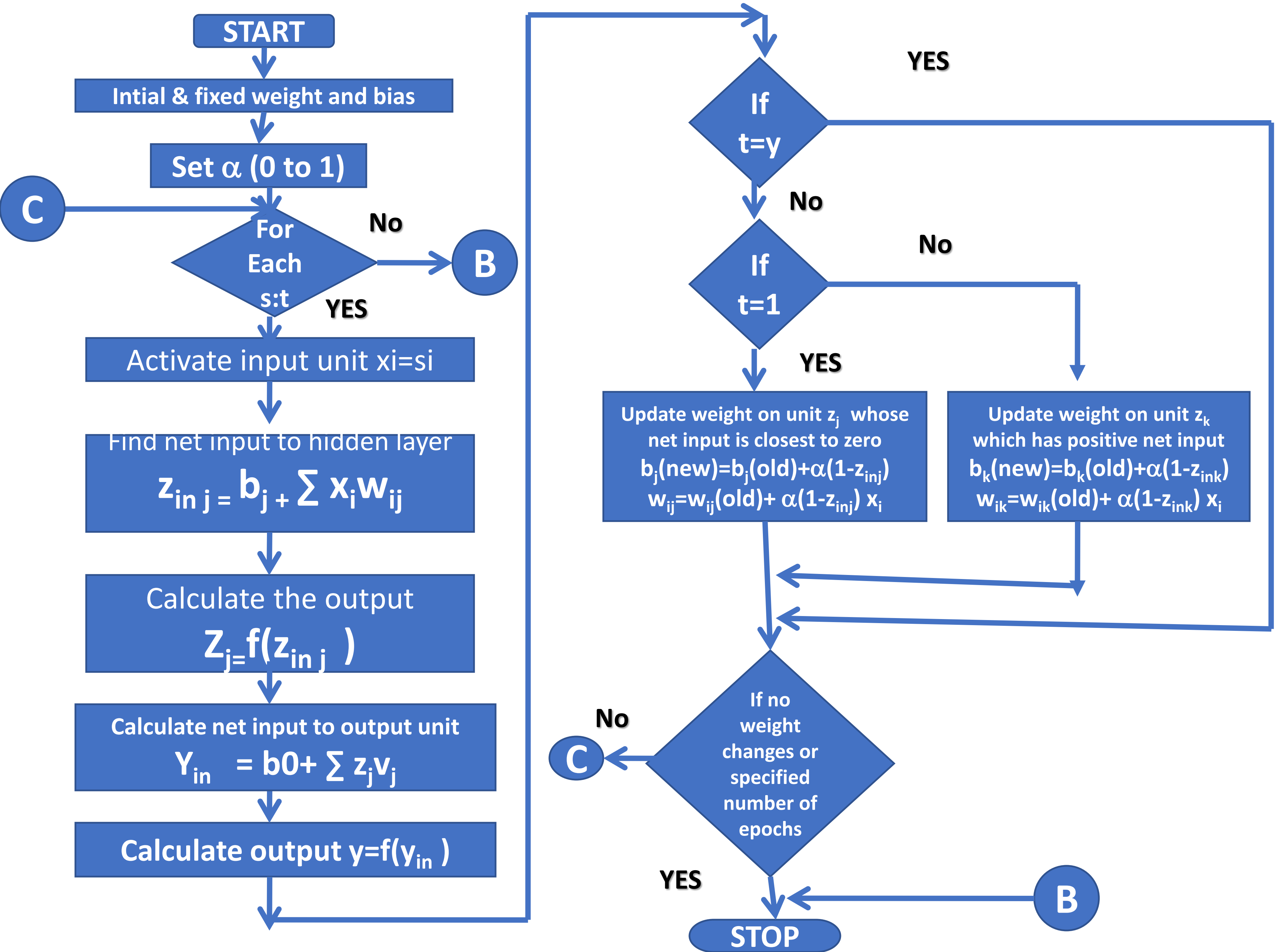
- Two or more adaline are integrated to develop madaline model
- Used for nonlinearly separable logic functions (X-OR) function
- Used for adaptive noise cancellation and adaptive inverse control
- In noise cancellation the objective is to filter out an interference component by identifying a linear model of a measurable noise source and the corresponding immeasurable interference.
- ECG, echo elimination from long distance telephone transmission lines

ARCHITECTURE

- ✓Simple madaline architecture consist of 'n' unit of input layer 'k' unit of Adaline layer and one unit of madaline layer
- ✓Each neuron in the adaline and madaline layer has a bias of excitation 1
- ✓Adaline layer is present between the input layer and madaline layer hence the adaline layer can be considered as hidden layer
- ✓The use of hidden layer gives the net computational capability which is not found in single layer net
- ✓But this complicate the training process


1
b1
x1
X1
w11
Z1
V1
w1j
1
w1m
wi1
1j
bj
b0
xi
Xi
wij
Zj
Vj
Y
wim
1m
Vm
wn1
wnj
bm
wnm
Zm
xn
Xn

TRAINING ALGORITHM



START

Intial & fixed weight and bias

Set α (0 to 1)

C

For Each s:t

No

B

YES

Activate input unit xi=si

Find net input to hidden layer

$z_{in\ j} = b_j + \sum x_i w_{ij}$

Calculate the output

$Z_j = f(z_{in\ j})$

Calculate net input to output unit

$Y_{in} = b0 + \sum z_j v_j$

Calculate output $y=f(y_{in})$

If t=y

YES

No

If t=1

No

YES

Update weight on unit z_j whose net input is closest to zero

$b_j(new)=b_j(old)+\alpha(1-z_{inj})$

$w_{ij}=w_{ij}(old)+\alpha(1-z_{inj})\ x_i$

Update weight on unit z_k which has positive net input

$b_k(new)=b_k(old)+\alpha(1-z_{ink})$

$w_{ik}=w_{ik}(old)+\alpha(1-z_{ink})\ x_i$

No

C

If no weight changes or specified number of epochs

YES

B

STOP

✓Only the weight between the hidden layer and input layer are adjusted

✓Weight for the output layer is fixed

✓The weight v1,v2,v3 .... vj and bias b0 that enter into the output unit Y are delimited so that the response of unit Y is 1

✓Thus the weight entering the Y unit may be taken as

$$v1 = v2 = \ldots vj = ½$$

✓Bias can be taken as

$$b0=1/2$$

✓The activation function of adaline and madaline unit are

$$f(x)= \begin{cases} 1 & \text{If } x >= 0 \\ -1 & \text{If } x < 0 \end{cases}$$

- **Step 0:** Initialize the weight. The weight entering the output unit are set. Set initial small random values for adaline weights. Also set initial learning rate α
- **Step 1 :** When stopping condition is false, perform step 2-3
- **Step 2:** For each bipolar training pair s:t, perform 3-7
- **Step 3:** Activate input layer unit. For i=1 to n xi=si
- **Step 4**: Calculate the net input unit to each hidden adaline unit

$$z_{in\ j} = b_j + \sum x_i w_{ij} \quad ,j= 1 \text{ to } m$$

- **Step 5:** Find output of each hidden unit

$$Z_{j=} f(z_{in\ j})$$

- **Step 6**: Find output of net

$$Y_{in} = b0 + \sum z_j v_j$$

$$y=f(y_{in})$$

- **Step 7:** Calculate the error and update the weight
  - A: if t=y , no weight updation is required
  - B: if t!=y , & t= +1 update weight on zj where net input closer to zero

$$b_j(new)=b_j(old)+\alpha(1-z_{inj})$$

$$w_{ij}=w_{ij}(old)+ \alpha(1-z_{inj})\ x_i$$

  - C: if t!=y and t =-1 update the weight on unit zk whose net input is positive

$$b_k(new)=b_k(old)+\alpha(1-z_{ink})$$

$$w_{ik}=w_{ik}(old)+ \alpha(1-z_{ink})\ x_i$$

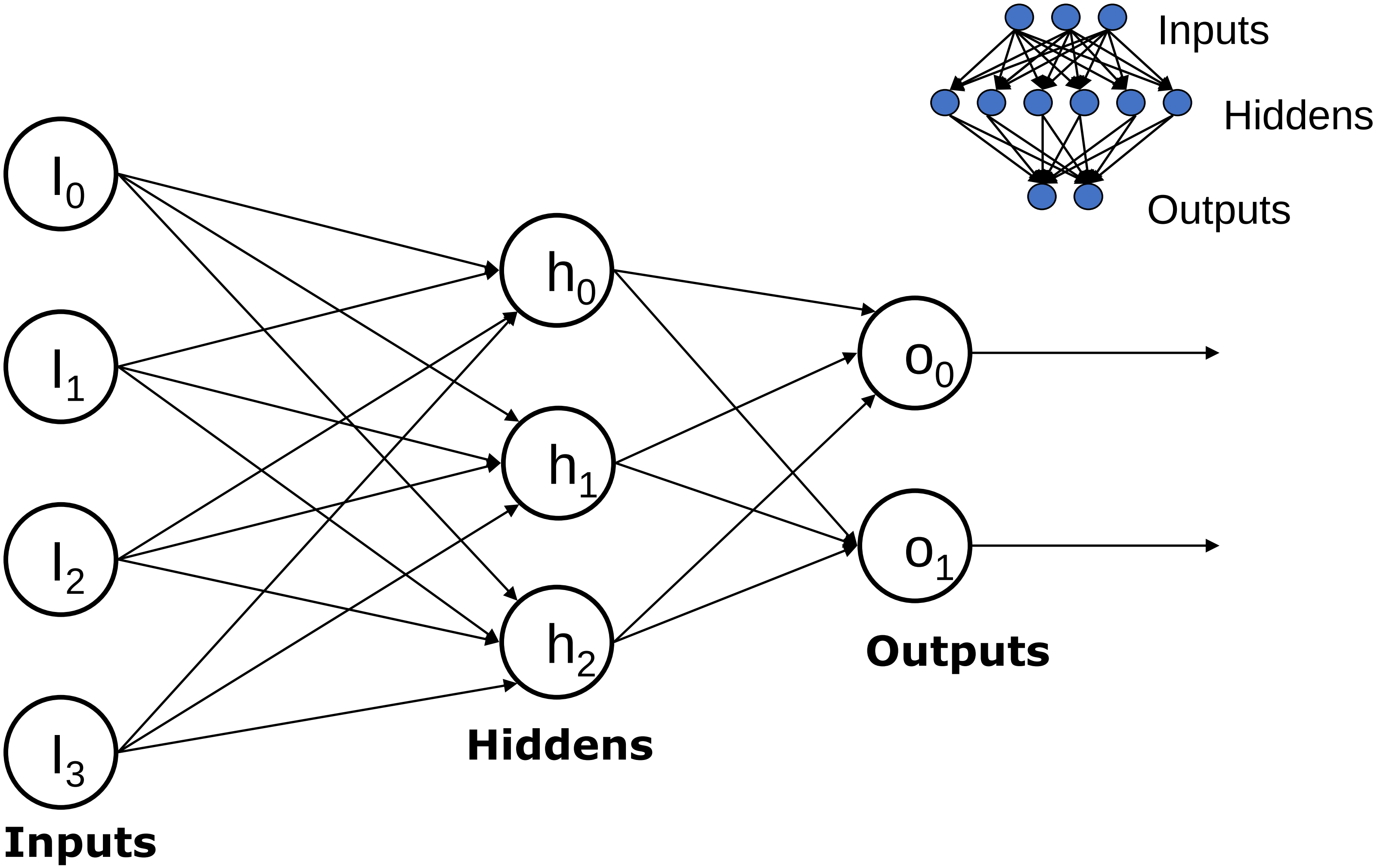
- **Step 8:** Test for stopping condition (if there is no weight change or weight reaches the satisfactory level, or if a specified maximum number of weight updation have been performed then stop or else continue

- A training procedure which allows multilayer feed forward Neural Networks to be trained.

- Can theoretically perform “any” input-output mapping.

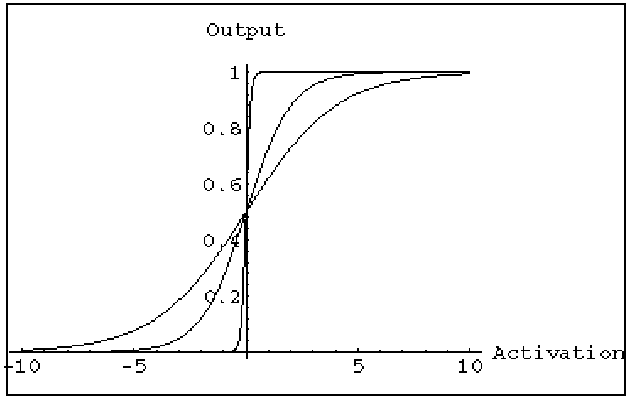
- Can learn to solve linearly inseparable problems.

MULTILAYER FEEDFORWARD NETWORK



MULTILAYER FEEDFORWARD NETWORK: ACTIVATION AND TRAINING

- For feed forward networks:
  - A continuous function can be
  - differentiated allowing
  - gradient-descent.
  - Back propagation is an example of a gradient-descent technique.
  - Uses sigmoid (binary or bipolar) activation function.



In multilayer networks, the activation function is usually more complex than just a threshold function, like $1/[1+\exp(-x)]$ or even $2/[1+\exp(-x)] – 1$ to allow for inhibition, etc.

GRADIENT DESCENT

- Gradient-Descent(training_examples, η)

- Each training example is a pair of the form <(x_1,…x_n),t> where (x_1,…,x_n) is the vector of input values, and t is the target output value, η is the learning rate (e.g. 0.1)

- Initialize each wi to some small random value

- Until the termination condition is met, Do
  - Initialize each Δwi to zero
  - For each <(x_1,…x_n),t> in training_examples Do

- ✓Input the instance (x1,…,xn) to the linear unit and compute the output o
- ✓For each linear unit weight wi Do

- $\Delta w_i = \Delta w_i + \eta \text{ (t-o) xi}$
- For each linear unit weight wi Do
- $w_i = w_i + \Delta w_i$

MODES OF GRADIENT DESCENT

- Batch mode : gradient descent

  w=w - η ∇ED[w] over the entire data D

  ED[w]=1/2Σd(t_d-o_d)2

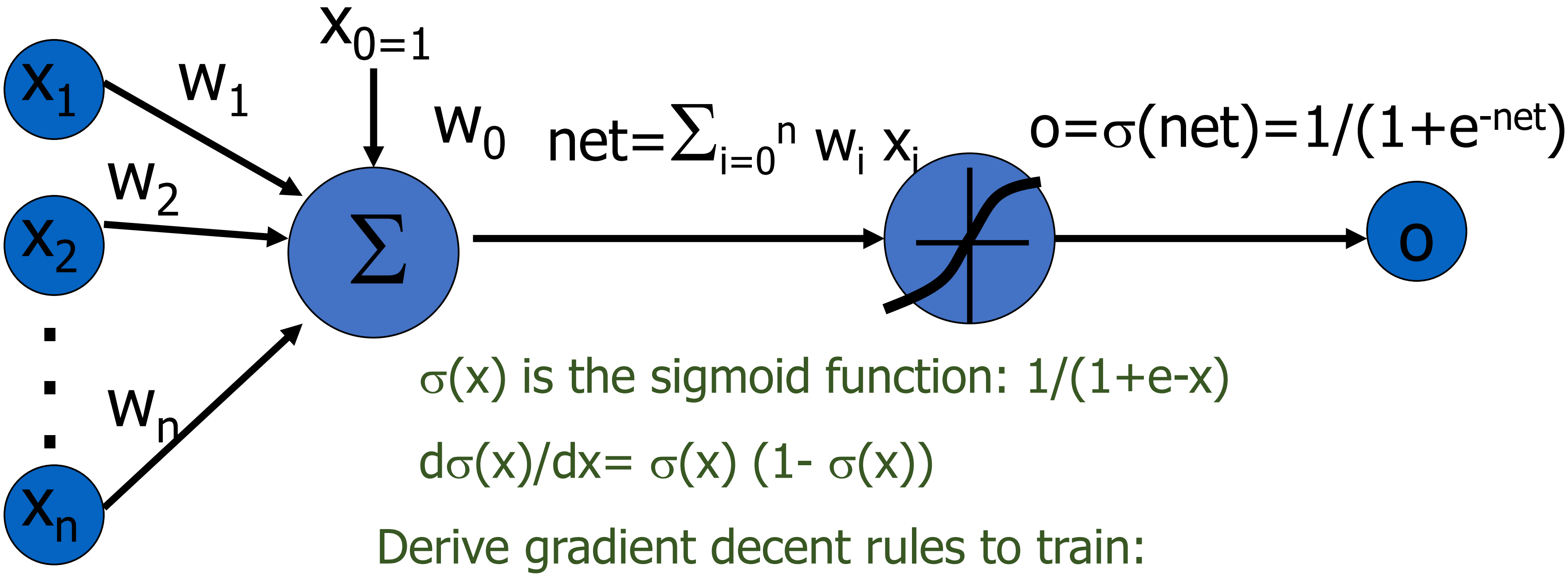
- Incremental mode: gradient descent

  w=w - η ∇E_d[w] over individual training examples d

  Ed[w]=1/2 (t_d-o_d)2

- Incremental Gradient Descent can approximate Batch Gradient Descent arbitrarily closely if η is small enough.

SIGMOID ACTIVATION FUNCTION



$net=\sum_{i=0}^{n} w_i\ x_i$

$o=\sigma(net)=1/(1+e^{-net})$

$\sigma(x)$ is the sigmoid function: $1/(1+e-x)$

$d\sigma(x)/dx= \sigma(x)\ (1- \sigma(x))$

Derive gradient decent rules to train:

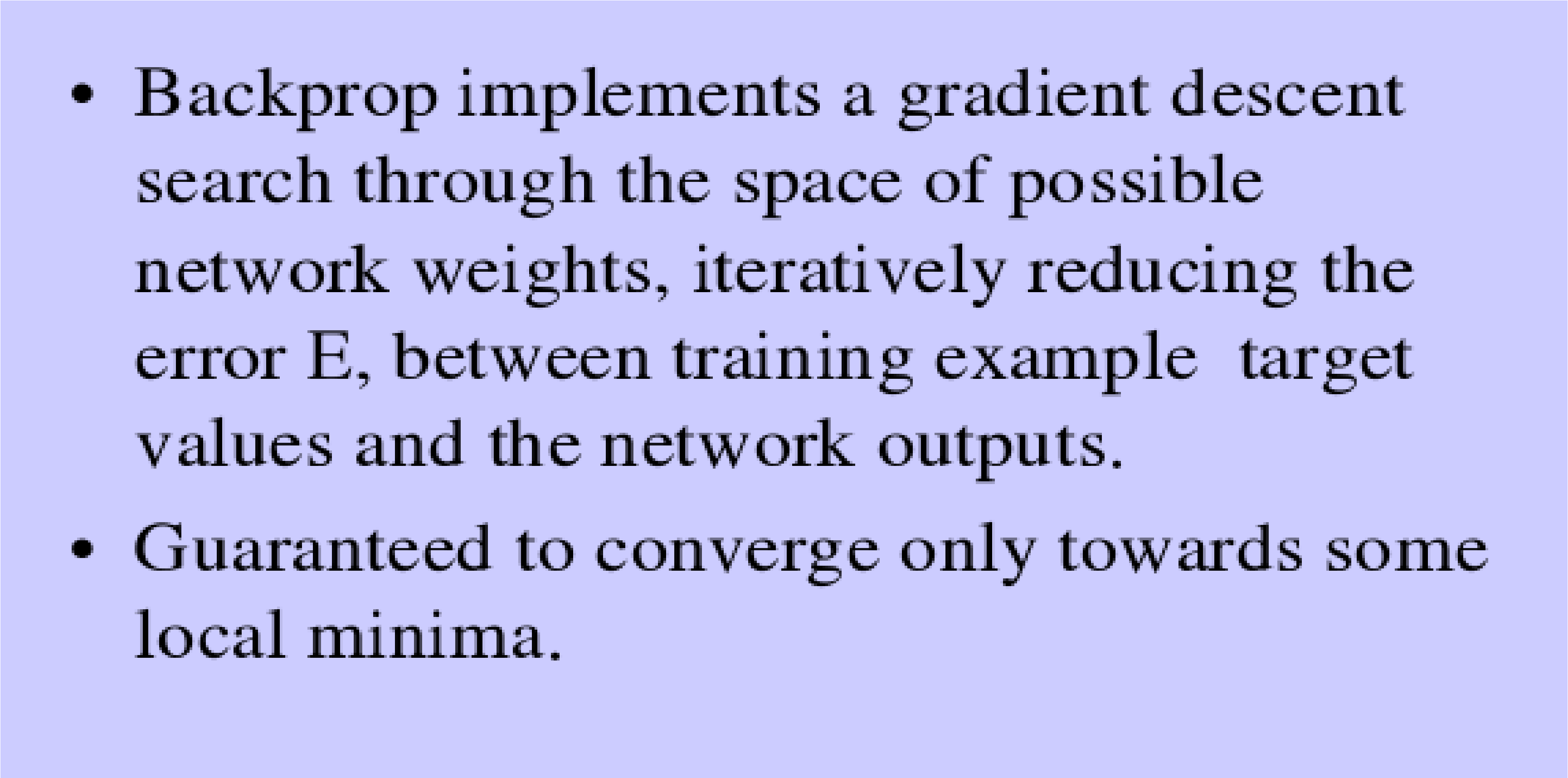
- one sigmoid function

  $\partial E/\partial w_i$ = $-\Sigma d(td-od)\ od\ (1-od)\ xi$

- Multilayer networks of sigmoid units

backpropagation

BACK PROPAGATION NETWORK



- Backprop implements a gradient descent search through the space of possible network weights, iteratively reducing the error E, between training example target values and the network outputs.
- Guaranteed to converge only towards some local minima.

BACKPROPAGATION

- Gradient descent over entire network weight vector
- Easily generalized to arbitrary directed graphs
- Will find a local, not necessarily global error minimum -in practice often works well (can be invoked multiple times with different initial weights)
- Often include weight momentum term

  $\Delta wi,j(t)= \eta\ \delta j\ xi,j + \alpha\ \Delta wi,j\ (t-1)$
- Minimizes error training examples
- Will it generalize well to unseen instances (over-fitting)?
- Training can be slow typical 1000-10000 iterations (use Levenberg-Marquardt instead of gradient descent)

- ✓Back propagation is a training method used for a multi layer feed forward network.
- ✓The network associated with Back propagation learning algorithm is called BACK PROPAGATION NETWORK
- ✓Processing elements with continuous differentiable activation functions
- ✓It is also called the generalized delta rule.
- ✓It is a gradient descent method which minimizes the total squared error of the output computed by the net.
- ✓Any neural network is expected to respond correctly to the input patterns that are used for training which is termed as memorization and it should respond reasonably to input that is similar to but not the same as the samples used for training which is called generalization.

- ✓The training of a neural network by back propagation takes place in three stages
  - ✓ 1. Feed forward of the input pattern
  - ✓2. Calculation and Back propagation of the associated error
  - ✓3. Adjustments of the weights
- ✓After the neural network is trained, the neural network has to compute the feed forward phase only.
- ✓Even if the training is slow, the trained net can produce its output immediately.

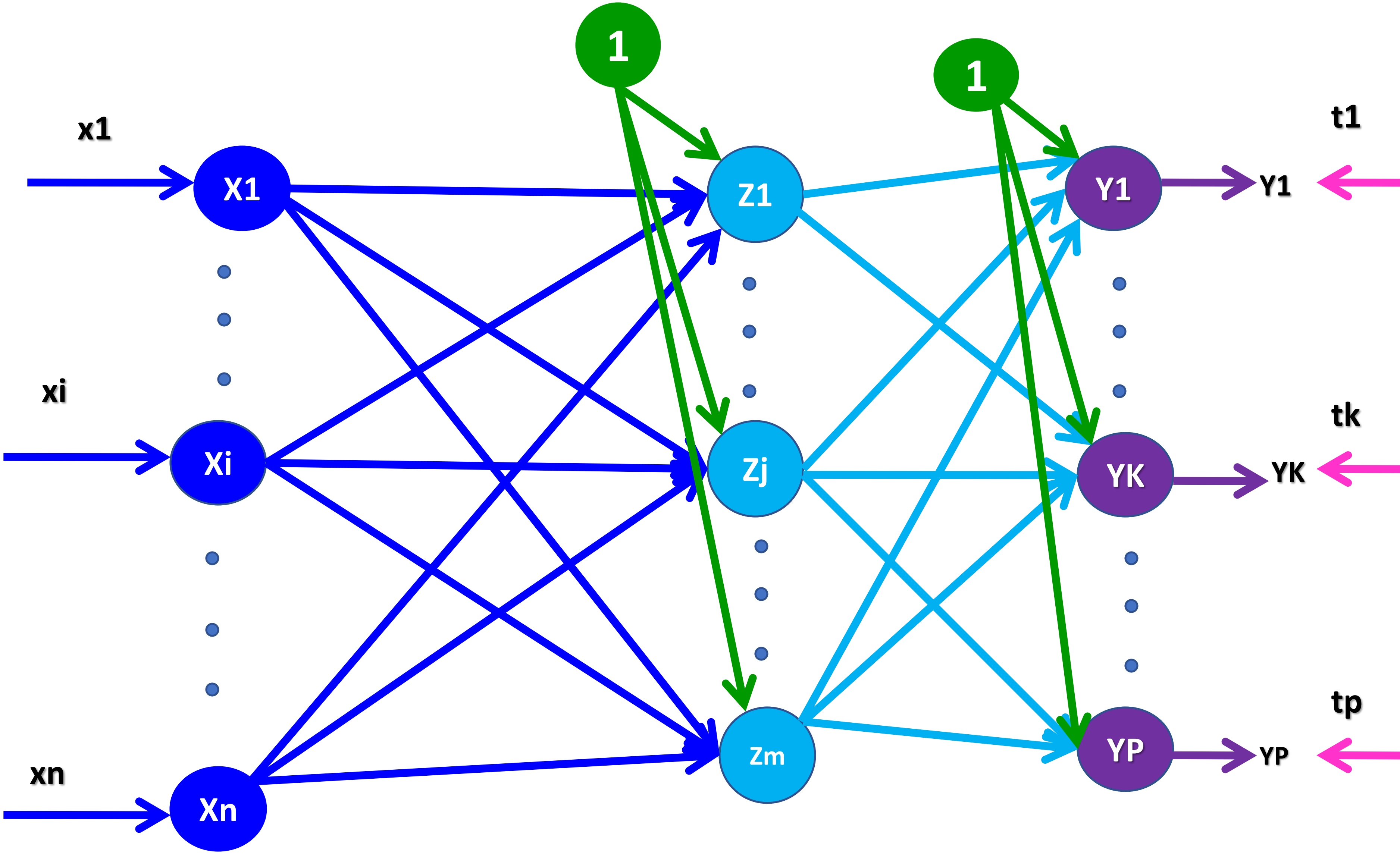
ARCHITECTURE

✓The output units and the hidden units can have biases.

✓These bias terms are like weights on connections from units whose output is always 1.

✓During feed forward the signals flow in the forward direction i.e. from input unit to hidden unit and finally to the output unit.

✓During back propagation phase of learning, the signals flow in the reverse direction.


1
1
x1
t1
X1
Z1
Y1
Y1
xi
tk
Xi
Zj
YK
YK
xn
tp
Xn
Zm
YP
YP

ALGORITHM

- ✓The training involves three stages
  - ✓1. Feed forward of the input training pattern
  - ✓2. Back propagation of the associated error
  - ✓3. Adjustments of the weights.
- ✓During feed forward, each input unit (Xi) receives an input signal and sends this signal to each of the hidden units Z1, Z2, ...Zn.
- ✓Each hidden unit computes its activation and sends its signal to each output unit.
- ✓Each output unit computes its activation to compute the output or the response of the neural net for the given input pattern.

✓During training, each output unit compares its computed activation yk, with its target value tk to determine the associated error for the particular pattern.

✓Based on this error the factor δk for all m values are computed.

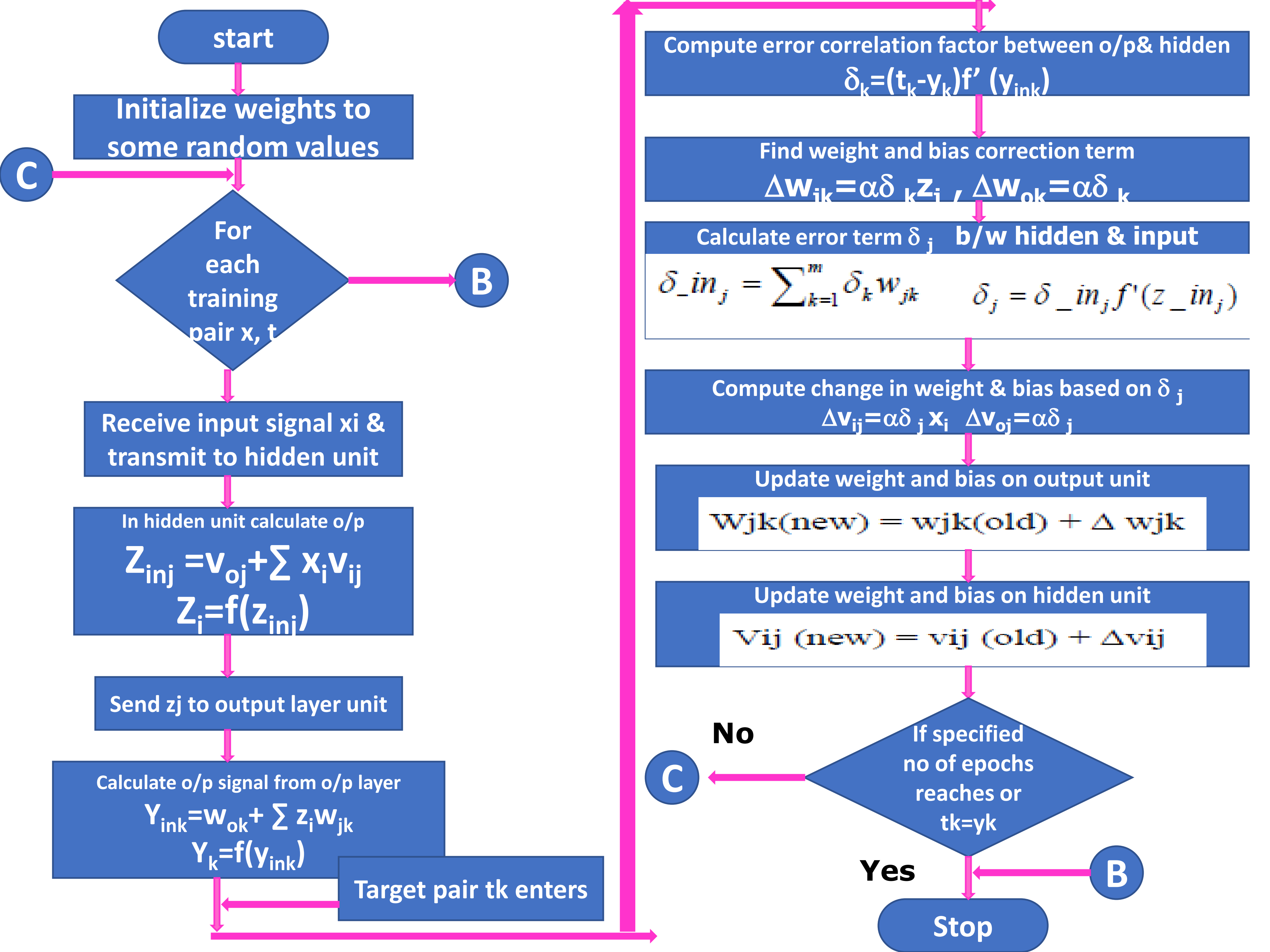
✓This computed δ k is used to propagate the error at the output unit Yk back to all units in the hidden layer.

✓At a later stage it is also used for updation of weights between the output and the hidden layer.

✓In the same way δj for all p values are computed for each hidden unit Zj.

✓The values of δj are not sent back to the input units but are used to update the weights between the hidden layer and the input layer.

- ✓Once all the ∂ factrs are known, the weights for all laye rs are changed simultaneously.
- ✓The adjustment to all weights wjk is based on the factor ∂k and the activation zj of the hidden unit Zj.
- ✓The change in weight to the connection between the input layer and the hidden layer is based on ∂j and the activation xi of the input unit



start

Initialize weights to some random values

C

For each training pair x, t

B

Receive input signal xi & transmit to hidden unit

In hidden unit calculate o/p

$Z_{inj} = v_{oj} + \sum x_i v_{ij}$

$Z_i = f(z_{inj})$

Send zj to output layer unit

Calculate o/p signal from o/p layer

$Y_{ink} = w_{ok} + \sum z_i w_{jk}$

$Y_k = f(y_{ink})$

Target pair tk enters

Compute error correlation factor between o/p& hidden

$\delta_k = (t_k - y_k) f'(y_{ink})$

Find weight and bias correction term

$\Delta w_{jk} = \alpha\delta_k z_j$, $\Delta w_{ok} = \alpha\delta_k$

Calculate error term δ_j b/w hidden & input

$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$ $\quad \delta_j = \delta\_in_j f'(z\_in_j)$

Compute change in weight & bias based on δ_j

$\Delta v_{ij} = \alpha\delta_j x_i$ $\quad \Delta v_{oj} = \alpha\delta_j$

Update weight and bias on output unit

Wjk(new) = wjk(old) + Δ wjk

Update weight and bias on hidden unit

Vij (new) = vij (old) + Δvij

If specified no of epochs reaches or tk=yk

No

C

Yes

B

Stop

ACTIVATION FUNCTION

✓An activation function for a back propagation net should have important characteristics.

✓ It should be continuous, Differentiable and monotonically non- decreasing.

✓For computational efficiency, it is better if the derivative is easy to calculate.

✓For the commonly used activation function, the derivative can be expressed in terms of the value of the function itself. The function is expected to saturate asymptotically.

✓The commonly used activation function is the binary sigmoidal function.

TRAINING ALGORITHM

- ✓The activation function used for a back propagation neural network can be either a bipolar sigmoid or a binary sigmoid.
- ✓The form of data plays an important role in choosing the type of the activation function.
- ✓Because of the relationship between the value of the function and its derivative, additional evaluations of exponential functions are not required to be computed.

- **Step 0:** Initialize weights
- **Step 1:** While stopping condition is false, do steps 2 to 9
- **Step 2:** For each training pair, do steps 3 - 8

Feed forward

Step 3: Input unit receives input signal and propagates it to all units in the hidden layer

Step 4: Each hidden unit sums its weighted input signals

Step 5: Each output unit sums its weighted input signals and applied its activation function to compute its output signal.

Back propagation

Step 6: Each output unit receives a target pattern corresponding to the input training pattern, computes its error information term

$$\delta_k = (t_k - y_k) f' (y_{\_ink})$$

Calculates its bias correction term

$$\Delta Wok = \alpha\delta k$$

And sends δk to units in the layer below

Step 7: Each hidden unit sums its delta inputs

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

Multiplies by the deriv ion to calculate its error information term

$$\delta_j = \delta\_in_j f'(z\_in_j)$$

Calculates its weight correc

$$\Delta vij = \alpha\delta jxi$$

And calculates its bias correction term

$$\Delta voj = \alpha\delta j$$

Update weights and biases

Step 8: Each output unit updates its bias and weights

$$Wjk(new) = wjk(old) + \Delta\ wjk$$

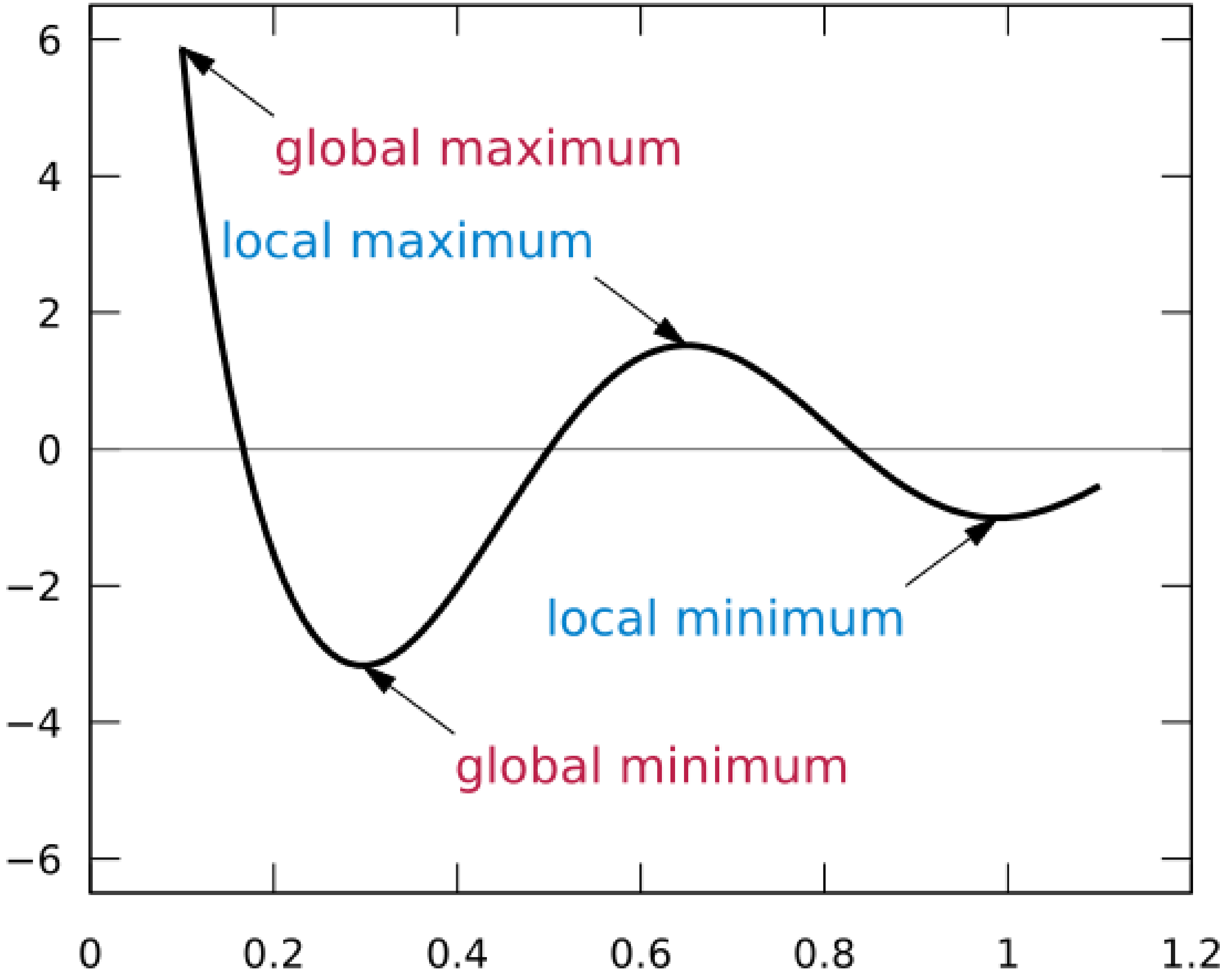
Each hidden unit updates its bias and weights

$$Vij\ (new) = vij\ (old) + \Delta vij$$

Step 9: Test stopping condition. The stopping condition may be certain number of epochs reached or when the actual output equal to target value

- ✓The above algo uses the incremental approach for updation of weight
- ✓That is weights are being changed immediately after a training pattern is present
- ✓There is another way of training called batch mode training where the weights are changed only after all the training pattern are presented
- ✓Batch mode requires additional local storage for each connection to maintain the immediate weight change
- ✓When a BPN used as a classifier, it is equivalent to the *optimal Bayesian Descriminator* function for assymptotically large set of training patterns

- ✓If the BPN algorithm converges at all , then it may get stuck with local minima and may be unable to find a satisfactory solution
- ✓The randomness of the algorithm helps it to get out of local minima
- ✓The error function may have large number of global minima because of permutation of weight that keep the network input output function unchanged


6
4
2
0
−2
−4
−6
0
0.2
0.4
0.6
0.8
1
1.2
global maximum
local maximum
local minimum
global minimum

LEARNING FACTORS OF BACK PROPOGATION NETWORK

- Training of BPN is based on the choice of various parameters
- Convergence of BPN is based on some important learning factors such as
  - Initial weight
  - The learning rate
  - The updation rule
  - Size and nature of training set
  - Architecture (number of layers and number of neurons in each layer

INITIAL WEIGHT

- The weights are initialized with some random values
- The choice of initial weight is determines how fast the network converges
- The initial weight cannot be very high becoz sigmoid activation function used here may get saturated from the beginning itself and the system may be stuck at the local minima
- One method for choosing weight is in the range

$$\left[\quad -3/\sqrt{o_{i,}}\ 3/\sqrt{oi} \quad\right]$$

- Where oi is the number of processing elements j that feed forward to processing element i

- The initialization can also be done by a method called **Nguyen- Widrow Initialization**
- Leads to faster convergence of network
- Improves the learning ability of hidden layer
- The random initialization of weight connecting input neuron to the hidden neuron is obtained by

$$Vij\ (new)=\gamma\ vij(0ld)/\ ||\ vj(old)||$$

LEARNING RATE α

- Affect the convergence of BPN
- A large value of alpha may speedup the convergence but may result in overshooting
- The range of alpha from 10^-3 to 10 has been used successfully for several backpropogation algorithms experiments
- Slower learning rate lead to slower learning

Momentum factor

- The gradient descent is very slow if the learning rate is small and oscillate widely if alpha is too large
- One method that allows a large learning rate without oscillation is by adding a momentum factor to the normal gradient descent method

GENERALIZATION

- A network is said to be generalized when it sensibly interpolate with input network that are new to the network
- When there are many trainable parameters for a given amount of training data, the network learn well but does not generalize well
- This is usually called over fitting or over training
- One solution is to monitor error on the test set and terminate the training when error increases
- With a small number of trainable parameters, the network fail to learn training data set to test the data set
- However computationally large number of nodes is capable of memorizing the training set at the cost of generalization
- As a result smaller net are preferred than larger ones

Number of Training Data

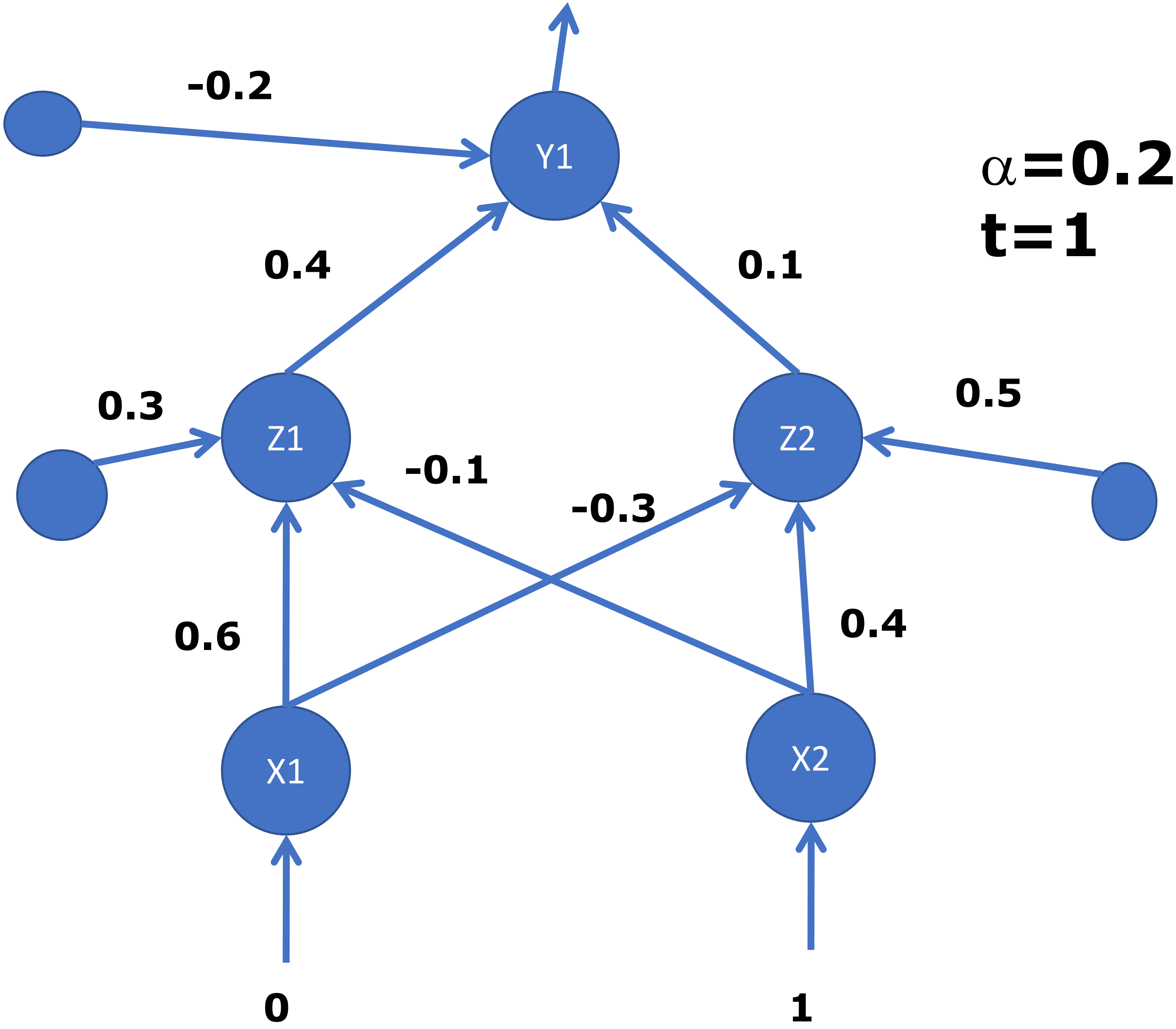
- Training data should be sufficient and proper
- There exist a rule of thumb which state that the training data should cover the entire expected input space, and while training , the training vector pair should be selected randomly from the set

Number of Hidden Layer nodes

- If there exist more than one hidden layer in BPN, then the calculation performed for a single layer are repeated for all the layers and summed up at the end
- In case of all multilayer feed forward network, the size of hidden layer is very important
- The number of hidden layer need for an application determined separately
- The size of hidden layer is determined experimentally
- For a network of reasonable size, the size of hidden layer has only be a relatively small fraction of input layer
- For example if the network does not converges to a solution , it may need more hidden layers

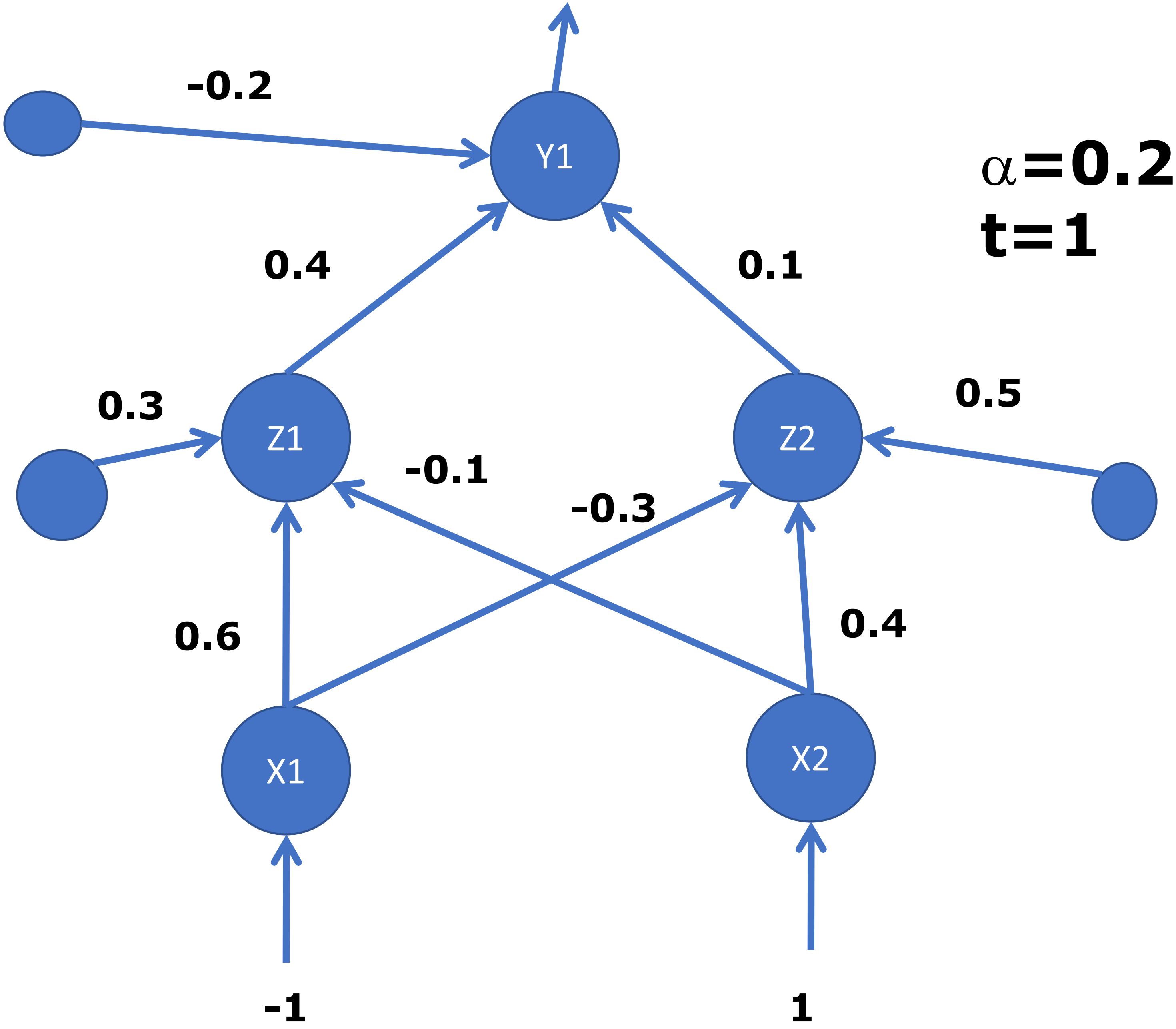
Testing Algorithm

- Step 0: Initialize weight. The weight are taken from training algorithm
- Step 1: Perform step 2- 4 for each input vector
- Step 2: Set the activation of input unit for xi
- Step 3: Calculate the net input to hidden unit and its output
- Step 4: Now compute the output for output layer unit. Use sigmoidal activation function



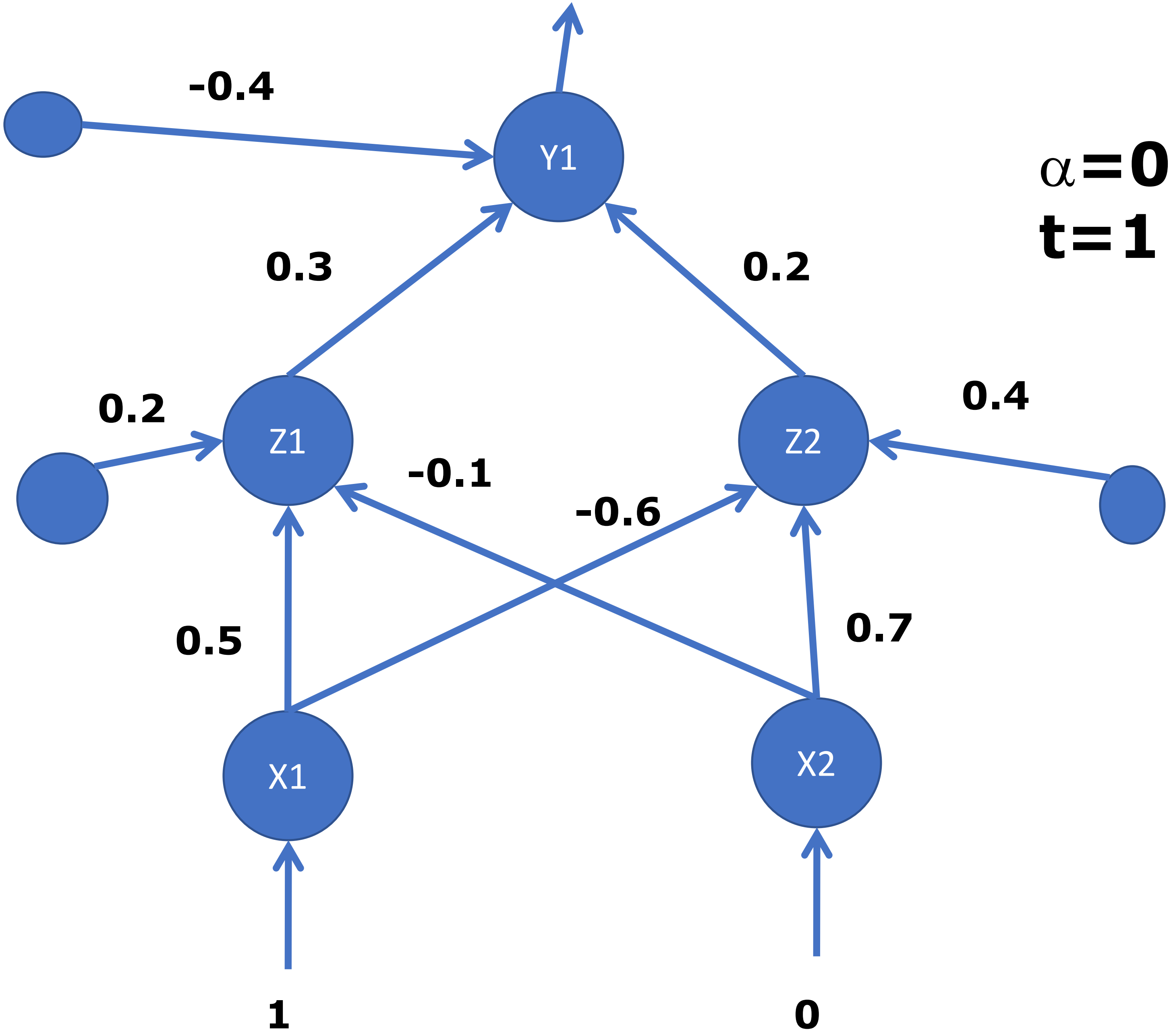
α=0.25

t=1



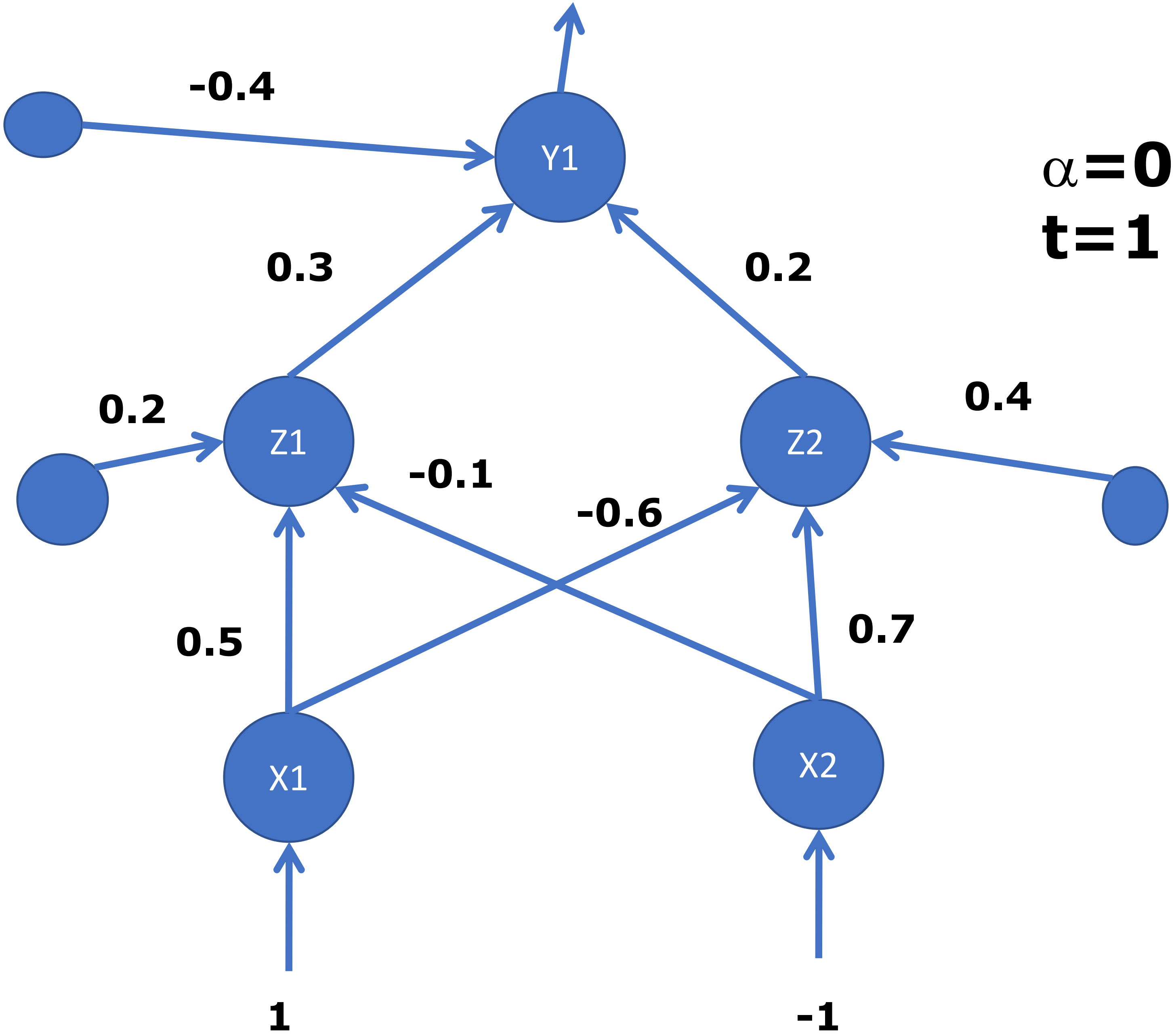
α=0.25

t=1



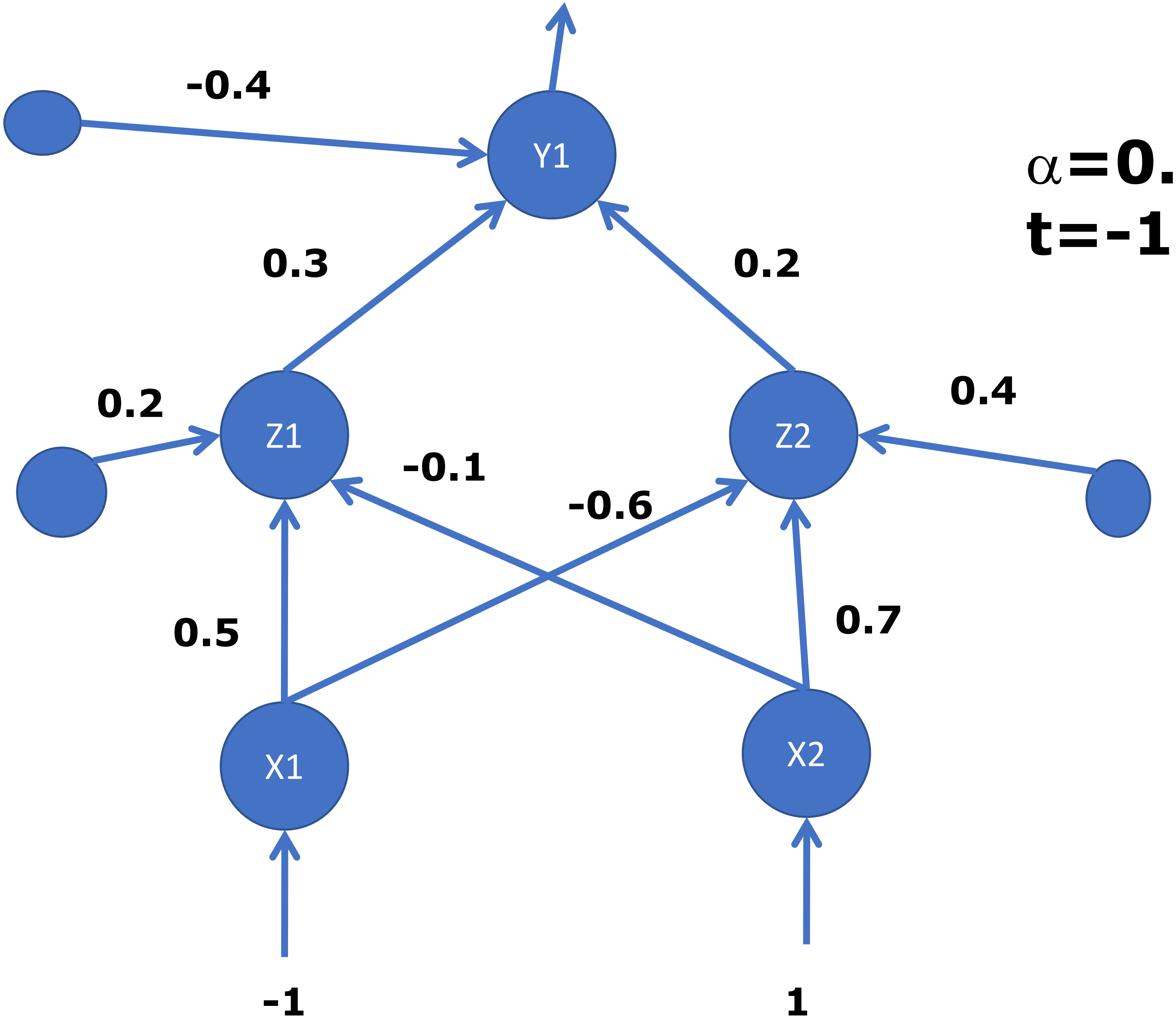
α=0.3

t=1



α=0.3

t=1



α=0.45

t=-1

APPLICATIONS OF BACKPROPAGATION NETWORK

- Load forecasting problems in power systems.
- Image processing.
- Fault diagnosis and fault detection.
- Gesture recognition, speech recognition.
- Signature verification.
- Bioinformatics.
- Structural engineering design (civil).